Faculty of Natural and Mathematical Sciences
Department of Informatics

King's College London
Strand Campus, London,
United Kingdom

7CCSMPRJ

Individual Project Submission 2023/24

Name:	Harsh Merchant
Student Number:	23033736
Degree Programme:	MSc Data Science
Project Title:	Enhancing Generative Tree Based Modeling With Differential Privacy
Supervisor:	Dr David Watson
Word Count:	12118

RELEASE OF PROJECT

Following the submission of your project, the Department would like to make it publicly available via the library electronic resources. You will retain copyright of the project.

☑ I **agree** to the release of my project

☐ I **do not** agree to the release of my project

Signature: 

Date: August 6, 2024

Department of Informatics
King's College London
United Kingdom

7CCSMPRJ Individual Project

Enhancing Generative Tree Based Modeling With Differential Privacy

---

Name: **Harsh Merchant**
Student Number: 23033736
Course: MSc Data Science

Supervisor: Dr David Watson

This dissertation is submitted for the degree of MSc in MSc Data Science.

Acknowledgements

I would like to thank my supervisor, Dr. David Watson, for his guidance and insights which has helped me in successfully completing this thesis. I am also very much thankful to my family, whose support and continued belief in me sustained my motivation throughout this journey. Additionally, I would like to thank all of my friends with whom I shared this adventure, making the process as memorable as the achievements. And to my beloved pets, Brownie and Luffy, thank you for the countless moments of joy.

To all of you, thank you for making this journey truly incredible and legendary.

Abstract

Synthetic data has become an essential tool across various domains, driving the need for effective generative models. Gradient Boosted Decision Trees and Random Forests are often used for the sake of synthetic data generation this is because they tend to give good results, are robust, and are effective. However, all of this, much like the generative models based on deep neural networks, pose strong possibilities of privacy breaches. These methods may potentially leak sensitive information and patterns from the original dataset into the synthetic data. To address this vulnerability, this thesis proposes integrating differential privacy into tree based generative models. This might allow for effective protection of privacy since it will retain useful information in the synthetic data with minimal risk of revealing individual data points. Differential privacy provides a firm foundation for privacy in data by ensuring that the privacy of individuals in the data is not significantly impacted due to participating in the database.

This thesis leverages the principles and methodologies of differential privacy and incorporates them into the adversarial random forest, a tree based generative model. The research successfully generates differentially private synthetic data through strategic injection of calibrated noise at critical points of the developed model. Thus, it augments differential privacy by a small factor in exchange for a little degradation of original data characteristics. This trade-off is required for the fulfilment of differential privacy

The results of the evaluations for the differentially private model over various datasets prove its effectiveness in privacy enhancement making it a promising foundation for further research into the application of differential privacy in tree based generative modeling. Thus, suggesting that a differentially private model based on trees may represent a feasible way to protect sensitive information while enabling valuable, data-driven insights and analysis. This work opens up a pathway for more secure and privacy compliant generation of data in many different domains.

Nomenclature

arfpy	Adversarial Random Forests with python
DNN	Deep Neural Network
dp arfpy	Differentially Private Adversarial Random Forests with python
DT	Decision Tree
EOGT	Ensemble Of Generative Trees
ϵ	Epsilon, a parameter for differential privacy budget
FORDE	Forests for Density Estimation
FORGE	Forests for Generative Modeling
GAN	Generative Adversarial Network
GBDT	Gradient Boosting Decision Trees
GDPR	General Data Protection Regulation
GF	Generative Forests
GT	Generative Trees
HIPAA	Health Insurance Portability and Accountability Act
IID	Independent and Identically Distributed
KDE	Kernel Density Estimation
KNN	K-Nearest Neighbors
MLE	Maximum Likelihood Estimation
RDP	Rényi Differential Privacy
RF	Random Forests
VAE	Variational Autoencoder

Contents

List of Figures

List of Tables

1 Introduction

Synthetic data generation is creating fake data to mimic the type of data that would be found in the real world. Normally, this is achieved with the use of different techniques like statistical modeling and machine learning. The generated synthetic data is purposefully designed to hold the properties of real data for using in applications such as testing, training, and analysis. It may range from helping to overcome data scarcity to mitigating bias and model training.

Over the past few decades, the generation of synthetic data has advanced quite a bit. Initially the interest was driven by big data, and later on, an even greater necessity emerged for generating large high quality datasets in machine learning and data analysis. Formerly, it was only applied under the context of simulations and tests, but its applications have already expanded to areas such as privacy preservation, algorithm development, and performance benchmarking. Deep neural networks have dramatically improved the field of machine learning, mainly concerning unstructured data, which comprises of images, audio, video, and text, where the traditional ways often fall short. This is especially evident in variational autoencoders [1] and generative adversarial nets [2], which excel at learning complicated patterns and producing new samples that closely resemble the original data points. Since the data is structured and often high-dimensional, it poses many challenges that can hinder the performance of deep learning.

In this regard, as a significant proportion of data from finance, healthcare, and business is tabular, and accurate data modeling and prediction are crucial in these fields, there is an imperative for producing samples that mimic the properties of such data accurately. To overcome this problem tree based methods are employed. The methods described are well suited to the aforementioned requirements of working with tabular data and give a very robust solution toward generating synthetic data while keeping the fundamental properties and relationships of the original datasets in place. Moreover, they need very little preprocessing and can automatically deal with missing values, which makes the task of data generation easier. Their ensemble nature and regularization techniques minimize the risk of overfitting, ensuring that the synthetic data generalizes well.

Sensitive information is always at stake when generating synthetic data, and its privacy raises concern on top of that. Privacy concerns have become increasingly important with the emergence of stricter data protection regulations and the growing awareness of data privacy issues. Traditional data anonymization techniques lack in many areas, and so this has provoked the development of advanced techniques such as differential privacy. This framework provides strong privacy guarantees, such that the privacy of any individual in the original dataset is protected. Hence, having potential to be a leading area of research in synthetic data generation.

1.1 Motivation and Aims

The need for relying on data increases as the world becomes data driven in more and more areas of society. This increases the demand for high quality datasets. However, synthetic data generation brings several risks with it , despite its many advantages. One of the major issues is related to the leakage of sensitive information from the original dataset into the synthetic data. This type of leakage can happen when the synthetic data retains some identifiable patterns or anomalies that do exist in the real data, and thus it might become possible for someone to make some inference about an individual's data. Such privacy breaches can have dire consequences.

In areas such as healthcare and finance, data privacy is crucial. The healthcare data mostly contain highly sensitive information about medical histories, conditions, and treatments of individuals. Therefore if this data is accessed by untrusted entities it may potentially lead to social stigmatization, loss of finances, and even threats to personal safety. Financial data is also said to contain important information on one's economic status, transactions, and individuals' credit history. Loss of privacy related to financial data may result in identity theft, financial impropriety, and substantial economic losses. Moreover, privacy of data in such domains is necessary not only for the sake of maintaining trust but also because of stringent regulatory requirements. For example, HIPAA in healthcare and GDPR in Europe.

This research is motivated by finding ways to overcome such privacy concerns while maintaining the good qualities of tree based models in data synthesis. Differential privacy offers a compelling solution by providing strong privacy guarantees through the introduction of controlled noise. By integrating differential privacy into tree based generative models, this research aims to create synthetic datasets, with the purpose of utility preservation while affording better privacy protection. Such datasets can then be used across various domains and in a variety of applications ensuring that there is safe and ethical usage of synthetic data. As a result, it would comply with with many data protection regulations, foster trust among data providers and users, and promote the broader adoption of synthetic data technologies.

1.2 Objectives and Summary

The primary objective of this research is to explore the effective integration of differential privacy into tree based generative models, specifically Random Forests. By doing so, it will try to alleviate privacy risks associated with synthetic data generation, and most importantly, it will retain the utility of the original data. This involves the assessment of the compromise reached between maximization of essential characteristics and patterns from the original dataset and minimization of individual's risk to ensure strong privacy protection through the use of synthetic data. Additionally, this research is going to compare the performance of the differentially private model with its non private counterpart on different metrics.

This report starts with the Background chapter, which defines some of the important concepts that the study stands on: Synthetic Data Generation, Tree Based Models, Privacy, and Federated Learning accompanied by previously published literature on this subject. The section Related Work reviews and critically evaluates existing research pertinent to the study. The chapter on Approach presents the experimental design, explaining how differential privacy is integrated with tree based models and the algorithms used, in addition to this it also explores alternative approaches concluding with the explaining of the experimental setup. The Results and Evaluation chapter presents and compares the conducted experiments and their results to show how effective is the proposed method. Conclusion gathers the main findings, discusses implications, indicates directions for further research, and reflects on the study's limitations. A detailed list of references is given at the end of the report. Additional detailed notes and results supporting the main findings of this report along with supplementary materials are included in the Appendix.

2 Background

2.1 Density Estimation and Generative Modeling

A probability distribution is an assignment of probabilities to each measurable subset of the possible outcomes of a random variable. For instance, test scores can be used to indicate how many individuals were scored in what numerical range of scores. In general, density estimation describes the process of estimating the said distribution by seeing how the values are spread out and then providing a smooth, continuous approximation for the underlying distribution. We approximate the underlying probability distribution since we do not typically know the exact form of the distribution from which our data samples come from. There are two primary methodologies for density estimation one wherein we assumes the form of a distribution for a set of given parameters. For example, When fitting the data to a distribution this could be a normal, exponential, or Poisson distribution the commonly used techniques are Maximum Likelihood Estimation (MLE). This is called parametric density estimation technique. On the other hand, non-parametric density estimation does not assume any form for the distribution. Instead, it estimates the density directly from the data. Examples include Kernel Density Estimation (KDE) which has wide applications and has become very popular in recent years.

To bridge the gap between understanding a dataset's distribution and generating new samples from it, we turn to generative modeling. Generative models use density estimation to find the joint probability distribution of the data. Consequently, they can create synthetic samples that are similar to the original data. Approaches to generative modeling can be taken from different directions like Gaussian Mixture Models which models data as a mixture of several Gaussian distributions, while latent variable models assume that the observed data was generated from hidden variables. Among these, Variational Autoencoders (VAEs) [1] which encodes data into a lower dimensional latent space and decodes it, thereby enabling the generation of new samples by drawing from this latent space and Generative Adversarial Networks (GANs) [2] that have a generator network, which makes up the data, and a discriminator network, which discriminates between the real and synthetic data. Both networks learn within a competitive setup until the generator produces highly realistic samples that the discriminator cannot distinguish between are particularly notable

2.2 Tree Based Models

Tree based models are an excellent class of machine learning algorithms that work on decision trees. Decision trees help in several ways given their ease of understanding, simplicity, and interpretability which are evident while performing the classification and regression task. They can work with both numeric and categorical data, picking up complex nonlinear relationships between features and the target. Notably, Random Forests and Gradient Boosted Decision Trees are widely popular tree based models because of their effectiveness and versatility.

Random Forests (RFs) [3] are one of the ensemble learning methods. They build multiple decision trees while maintaining an ensemble learning technique that aggregates the results to make a prediction. The technique not only helps in improving the robustness of the model in accuracy but also reduces the overfitting risk, which may come in individual decision trees. In the training phase, Random Forests develop many different decision trees such that each is trained on a random subset of the data and random features. This will effectively add diversity among the trees and inject power to increase stability and performance of the model. For classification, the Random Forest will output the class mode suggested by all the trees. In a regression problem, the prediction is given as the mean of all the trees. Averaging over the decision trees smoothens down the variance and actually augments its generalization. Moreover, another reason for the popularity of Random Forests is that they can handle enormous data sets effectively and perform well when there is missing information for a dataset. Being able to give insights about which features are important adds to interpretability and allows practitioners to understand better which features contribute the most toward the predictions.

In contrast, Gradient Boosted Decision Trees (GBDTs) [4] is a type of sequential ensemble method, meaning that trees are built one after the other. Each of the new trees is fit to correct the errors of all the previous trees, but now the target of the modeling is on the residuals (differences between responses in the observed and the previous model). In other words, boosting allows GBDTs to build a strong predictive model out of a series of weak learners (decision trees), each taking care of different aspects of the data. These are highly flexible, ensuring a good predictive accuracy in almost all kinds of tasks, be it regression, classification, or ranking. They basically operate by optimizing a loss function, measuring the model's capability in performing better or poorly. The most popular versions currently are XGBoost [5] and LightGBM [6]. Each of them has speed, efficiency, and the capability of handling data that is strictly better than others. GBDTs can capture complex interactions between features and tend to achieve high accuracy. However, GBDTs easily overfit in comparison with RFs, generally requiring careful tuning of hyperparameters. Overall, both models are very suitable for synthetic data creation due to their abilities to model complex data distributions and intrinsic robustness.

2.3 Privacy

The idea of using data in machine learning really raises a critical concern which is how do you ensure sensitive information such as personal health records, financial transactions, or any other personally identifiable information are not leaked. Privacy guarantees that these individual's data are not accessed without their consent and further prevents the inference of sensitive information. Traditional approaches to privacy involve anonymization and aggregation, which make it hard to recognize the individual data points. Yet, such techniques often fail to provide strong privacy guarantees since sophisticated re-identification attacks are sometimes able to reveal the underlying data.

Several types of identifiers can reveal a person's identity like:

- **Direct Identifiers**: Information that directly identifies a person, for instance, name of an individual, social security number, and email address.
- **Indirect Identifiers (Quasi-Identifiers)**: Information such as age, gender, and ZIP code when combined with other data, indirectly identify an individual.
- **Sensitive Attributes**: Certain pieces of information such as health status, financial records, and personal preferences, are held at an individual level, which should not be related back to a person and are private.

There exist attacks that potentially exploit the above information from datasets to learn about a individual's private details. The reconstruction attack tries to reconstruct original data from aggregated or anonymized datasets. Differential attacks are those that exploit subtle differences in the response to the queries on a dataset. Even in aggregated data, if an attacker can issue enough queries and see small changes in the responses, they can infer whether or not the data of a particular person exists there, thus breaking the privacy. There is also inference attacks that attempt to deduce sensitive information about individuals or specific attributes across a dataset even when such information is not explicitly available. These attacks use data that is in the public domain or any other form of information retrievable to infer more facts. For instance, an attacker might use background knowledge such as demographics.

Generative AI models can create high-quality synthetic content but are often trained on sensitive data. A set of studies shows that indeed, in such models like GPT-2, there is the possibility for generating identity revealing text, upon seeding with particular prompts. Thus when not sufficiently protected, it inadvertently reproduces and leaks private details from its training datasets. The article [7] lists several key privacy threats and attacks like the ones mentioned above and also proposes various mitigation strategies like data sanitization, obfuscation, differential privacy, deduplication, replication detection, and machine unlearning though it does not provide a deeper technical analysis, methodology and evaluation of solutions effectiveness.

2.4 Federated Learning

Federated learning is a novel machine learning paradigm that combats the privacy issue by enabling multiple parties to collaboratively train a model without sharing data. With centralized learning data from various sources is aggregated into a single location for model training which poses significant privacy risks. Federated learning, on the other hand, enables the training of a global model while distributing learning across devices or servers, each holding local data. The model parameters are then updated locally and collected centrally so that the raw data remains decentralized and private.

Some benefits to federated learning include protection of user's privacy and their data by avoiding an aggregation of the raw data thus preventing data-breaches and unauthorized access to information. There is also increase in scalability since the computational power of many devices can be employed to run complex models over large scale datasets. Personalization is another benefit in which local models can be fine-tuned in order to adapt to user or organizational data in a manner that avails more personalized and relevant predictions. On the other hand, federated learning brings additional challenges, such as managing communication overhead, the guarantee of model convergence, and dealing with non-IID (Independent and Identically Distributed) data over different devices.

3 Related Work

3.1 Deep Neural Networks vs Tree Based Models

The works of Grinsztajn in [8] and Borisov in [9] are dedicated to reviewing the weak points of Deep Neural Networks (DNNs) in tabular settings. They compare them to tree based methods like XGBoost and Random Forests. Both providing extensive benchmarks and highlighting relevant factors like data characteristics, hyperparameter tuning, and representation power.

[8] argued for tree-based models due to their robustness and interpretability, highlighting their superiority in handling medium sized tabular datasets. They also outline the three main challenges for tabular specific neural networks firstly the robustness to uninformative features, Secondly the preservation of data orientation, and lastly the capability of learning irregular functions. [9] acknowledges the power in tree-based models but finds a lot of potential with DNNs, when appropriately preprocessed and engineered to have adequate features.

Both papers concurred on one point which is research has to be done to develop models that can handle the challenge of high dimensionality and sparsity of tabular data. They also emphasize the need for a standard benchmark against which we can measure learning from tabular data, so that evaluation methodologies do not vary and results across different works can be compared. [8] found that tree-based models remain to be state-of-the-art in medium sized tabular datasets, even without considering faster training times than deep learning algorithms. They thus provide a new benchmark and baselines, allowing testing of new algorithms within a fixed budget of hyperparameter optimization. They also probe for inductive biases that differ between tree based models and neural networks, with the interpretation that neural networks fail at learning irregular patterns and their rotation invariance hurts performance on tabular data. The state-of-the-art deep learning methods for tabular data did not perform much better than the use of tree based models on newly created benchmarks depicted in [9]. This emphasizes the importance of benchmarks to ensure new models generalize well across diverse datasets. These papers have led this research to pursue tree based models instead of Deep Neural Networks for generative modeling.

3.2 Generative Trees and Forests

In their paper [10] authors articulate the issues of DNNs at hand and then put forth a new proposition, in which models based on decision trees can be used for generative tasks. This is germane to the current research insofar as it pushes further on the use of tree based models in contexts of generative models. Thus, opening up promising paths for integrating differential privacy within tree based generative models. [10] proposes Generative Trees (GTs) to try to imitate beneficial properties of Decision Trees (DT) for tabular data classification. The authors proposed an adversarial training algorithm compliant with boosting for GTs, which tackles the generative modeling of tabular data through strengths of induction of decision trees. It includes the properness condition of a

loss function so that it should reflect correctly the true nature of data and model predictions. For example, via proper scoring rules such that the model is properly calibrated on the supervised loss. This results in a variational GAN style loss formulation, which incorporates principles from variational methods optimized under calibration conditions satisfied by DTs. In simpler terms the method leverages a variational measure based divergence from the discriminator side to derive the loss for the generator, which is tight characterized in the case of the discriminator satisfying the calibration property. It gives rise to a unified loss function to train the generator. They then proposed two training algorithms for GTs: a standard adversarial training and a "copycat" training procedure. In the copycat method the generator replicates the discriminator's tree structure and thus, makes training the discriminator very easy, thereby allowing for convergence. This method can be more effective than traditional adversarial training and is simple and is designed with clear, direct guarantees of convergence. The fine grained study on well defined loss functions in their relation to training generative models gives some clues that helps implement differential privacy in a fine tuned manner in this class of models. Furthermore, the copycat training scheme sets up a new high standard regarding efficient training mechanisms that could be potentially adopted for the improvement of privacy preserving generative modeling.

In the follow up paper, [11], the authors continue their work on tree based generative models and tackle some of the challenges that arise during the generation of tabular data. This work builds on their prior work in the field of generative trees and presents new techniques and algorithms to provide better generative models in tabular data. Their main aim here is to introduce Generative Forests (GF) consisting of sets of trees so as to enhance density modeling and data generation for tabular data. They argue against the current approaches for three reasons, the difficulty in maintaining an exact sampler with respect to the underlying density, the challenge of developing powerful generative models for tabular data, and the conceptual gap between supervised learning algorithms such as boosting and their application in data generation. Their proposed solution, GF.BOOST, eases the training process by using a supervised training scheme that can be easily incorporated into popular algorithms for the induction of decision trees with minimal modifications. The architecture of the model has been elaborated, highlighting the combinatorial benefits of having more than one tree over having one tree. All the trees are being used to produce each observation. Further, it introduces Ensembles Of Generative Trees (EOGT) as an alternative to GF being more memory-efficient than the latter while preserving combinatorial properties. Experimental results show that the proposed generative forests are competitive and sometimes even outperform the state-of-the-art methods like MICE for missing data imputation and Advarsarial Random Forest for data realism.

Overall, While the contributions of authors to the work in [10] and [11] make for very fertile ground for further elaborating more potent and scalable generative models for tabular data. There is the privacy aspect that is lacking in the synthetic data generation methods proposed in both the papers. It is with these identified privacy gaps in mind that this research aims to extend these contributions to developing secure

and privacy compliant synthetic data generation methods while maintaining high data utility.

3.3 Adversarial Random Forests

The paper [12] introduces a novel approach to density estimation and data synthesis using an unsupervised form of random forests inspired by generative adversarial networks. It builds from a recursive procedure in which the trees in the forest learn structural properties of the data through generation and discrimination phases. It proves to be consistent under minimal assumptions and is especially optimized to work efficiently on tabular data with mixed attributes.

Initial Framework with Iterative Improvement: adversarial random forest starts by building an initial framework of a random forest. This framework is then trained to classify data into real and synthetically generated classes. The synthetic data at this stage is formed by simple random sampling from the marginal distributions of the data attributes. This initial setup creates a fundamental adversarial setting, where the forest's objective is to improve its ability to classify data points correctly as either real or synthetic.

Recursive partitioning and local independence: Recursive partitioning is key to the ARFs framework, such that every tree of a forest tries to partition data space in such a way that data within each leaf node becomes locally independent. This is measured by the independence criterion inside the leaf, for which the intent is that the joint distribution of the data points must factorize into the product of the marginal distributions. The factorization is critical as it eases the difficulty of density estimation from a possibly complex multivariate problem to simple univariate distributions within each partition.

Coverage and Refinement: In the process of iterative refinement, the coverage metric is critical in quantifying the percentage of data covered by each leaf. After the initial training phase, the ARF verifies the characteristics of the coverage and independence of each leaf to learn where the model's performance can be improved. Leaves that are poorly covered or whose variables are highly dependent on each other are then focused more on the second iteration when the partitions are more refined.

Density Estimation Approach in FORDE: For each of these segments of interdependent subgroups of leaves, the data is assumed to be more homogeneous, thereby making it much easier to estimate density. The technique focuses on estimating conditional distributions within each leaf, assuming that variables within these partitions can be treated as independent. The assumption of independence within the leaves allows one to consider applying much simpler statistical methods, such as Gaussian models for continuous data or categorical distributions for discrete data for density estimation. Lastly, it aggregates density estimate from all leaves taking into consideration the coverage so as to model the overall density of the dataset. Thus, to provide accurate density estimates, each segment's contribution is weighted based on its prevalence in the dataset.

FORGE's Synthetic Data Generation: FORGE creates new synthetic data points that resemble the learned distributions given the density estimates for leaves.

This is achieved by sampling from the estimated models for each leaf. Since these models are simplified and assume local independence, generating the new points can be carried out effectively without losing the statistical properties of the data in any given segment. In order to accomplish this, FORGE will aggregate samples across a number of leaves such that any properties and distributions learned by the forest can be taken into account to ensure that the synthetic data represents the entire dataset as closely as possible. The aggregated method does not only guarantee that diverse data samples are taken from the spectrum of the original data's characteristics but also guarantees that the complex multivariate relationships present in the original data are retained in the generated synthetic dataset.

In summary, ARF's flexibility and efficiency in handling tabular data make it a prime choice for integrating differential privacy mechanisms, a feature that is not fully explored in [12]. Thus establishing a strong basis for creating potent differential private generative models.

3.4 Differential Privacy

The seminal work on "Differential Privacy" [13] by Cynthia Dwork has established a foundational framework for constructing privacy preserving data analysis methodologies.

[13] gives proof showing that adversary having access to auxiliary information along with the access to database with the use of privacy mechanism models like non interactive(Sanatized version of the dataset is provided) and interactive(Interface for users to query the database is provided) allows the adversary to make a privacy breach. Differential privacy makes it such that "Any disclosure will be within a small multiplicative factor just as likely whether or not individual is in the database. Thus there will only be negligible to small increase in the risk to an individual in participating in the database and nominal gain by concealing one's data." [13]

In her work, she provides a precise mathematical definition of differential privacy along with strong mechanisms for maintaining individual privacy while allowing data to still be analyzed in a meaningful way. The core idea of Differential Privacy is that the "The risk should not substantially increase as a result of participating in the database." [13]

According to [13], A randomized function K provides ϵ-differential privacy for all datasets D and D' differing by at most one element, for all subsets of outputs S: This concept is mathematically formalized by the inequality:

$$\Pr[K(D) \in S] \leq e^{\epsilon} \Pr[K(D') \in S] \tag{3.1}$$

Here's a breakdown of the terms and their significance:

- K represents a randomized algorithm or function that processes the dataset and gets an output.
- D and D' are two datasets that differ in exactly one data point. This means D' can be obtained by adding or removing one individual's data from D.

- S is a subset of possible outputs that the function K might produce.
- $\Pr[K(D) \in S]$ denotes the probability that the function K will produce an output within the subset S when run on dataset D.
- ϵ (epsilon) is a non-negative parameter that quantifies the privacy loss. Smaller values of ϵ correspond to stronger privacy guarantees. It controls the trade-off between privacy and accuracy.

The paper introduces mechanisms for adding noise such as the Laplace Mechanism for numeric and Exponential Mechanism for non-numeric queries.

In [13], Dwork considers the use of Auxiliary information that is when the adversary may have some additional external information. It makes sure that sufficient privacy guarantees are made regardless of such knowledge, which is an important practical requirement due to the rich multifaceted nature of data contexts. Differential privacy is also safe against queries that are adaptive, wherein the type of query to be asked and its parameters rely on the output of previous analyses and queries. Dwork's framework allows prevention of these inquiries from exploiting the differential privacy mechanisms, hence shielding privacy through these sequences of requests for interlinked data.

Dwork's principles offer a very good guideline when combining differential privacy into tree based generative models such as Random Forests and Gradient Boosted Decision Trees. By doing this, the models can make use of the properties of differential privacy by adding noise during model training or data generation steps thus ensuring that it does not reveal any sensitive information about any single individual, while also making sure that synthetic datasets produced still maintain the properties of original data. This enhancement will serve the applicability of these models in sensitive applications and align with ever growing expectations and regulations over data privacy.

3.5 Differential Privacy in Generative Adversarial Network

The paper [14] sheds further light on integrating differential privacy into the architecture of Generative Adversarial Networks. Providing a significant solution to problems in generative models where potential information can be leaked out of sensitive training data, especially when such models are applied to private or sensitive datasets like medical records.

The paper firstly makes use of differential privacy to protect individual data points within the training set. The intuition behind their differential privacy is that adding one data point and removing one data point should not dramatically change the result of a GAN. This is achieved by adding noise to the gradients during the training process. Particularly it is added according to the moments accountant method, which helps in tracking the privacy loss over multiple training iterations and ensures that the overall privacy budget is maintained.

Their framework is built upon the Wasserstein GAN (WGAN). Based on the Wasserstein GAN (WGAN) framework, the distance of Wasserstein provides a more stable and

meaningful measurement for the difference between real and generated data distributions than the Jensen-Shannon divergence in traditional GANs. Gradient clipping, and Gaussian noise injection are used together to enforce the Lipschitz constraint in models so as to keep it stable during training and ensure convergence.

The authors present theoretical justifications for privacy guarantees, in addition to demonstrating the efficacy of their technique by comprehensive empirical testing on benchmarks such as MNIST and MIMIC-III. All these techniques and principles described in this study can be used to make sure that the synthetic data generated by tree-based models also preserves privacy. The fact that the DPGAN framework has already been applied successfully to sensitive data, such as medical records, illustrates both the importance and possibility of these privacy preserving techniques.

One of the strengths of [14] is that the theoretical foundation has been rigidly laid and deep empirical validation is done. The guarantees in terms of privacy are pretty strong since the use of Wasserstein distance and the moments accountant method brings good privacy, while maintaining utility in the produced data. Thus, This also shows us that the complexity of implementing differential privacy in both deep neural networks and tree-based models is high. Additionally, the impact of noise on the quality of generated data, especially in high-dimensional settings, remains a challenge that needs careful consideration. This can make it somewhat more complex to implement differential privacy. Moreover, the limitation in the usage of Neural Networks for tabular data, as presented before remain a bottleneck.

3.6 Differential Privacy in Federated Boosted Decision Trees

The paper [15] tackles the emerging requirement for scalable, safe, and efficient privacy preserving machine learning models capable of being trained over distributed data. This study examines Gradient Boosted Decision Trees (GBDTs) under Differential Privacy (DP) restrictions and in federated learning contexts. In this proposal, they present new methods to gain high utility and provide statistically stronger privacy guarantees by exploiting the flexibility of Rényi Differential Privacy (RDP), a recent refinement in differential privacy that allows more fine grained analysis by considering moments of the privacy loss distribution.

The proposed approach in the paper decomposes the GBDT algorithm into five main components, and each of them is adapted to the federated setting while satisfying RDP. The Split Method combines histogram based methodologies for scalable privacy preserving decision making while adding randomness using Partially Random(PR) and Totally Random (TR) strategies to improve privacy. Secondly, Weight Updates apply Averaging, Gradient-based, and Newton-based methods to optimize the model with gradients and Hessians. Then Split Candidate Generation independently generates candidates based on uniform distribution, and Iterative Hessian splitting enhancing data representation while enforcing privacy. After which Feature Interactions are managed by limiting the scope of data used in each decision, enhancing interpretability and privacy. Lastly, Batched Updates reduce the communication demands by collating updates to improve data efficiency and accelerate convergence.

While the authors primarily aim to improve privacy for decision making models in federated environments, the addition of generative modeling could vastly expand the utility of their framework. Generative modeling is particularly well suited to federated settings for several excellent reasons. Firstly, it facilitates data augmentation, allowing for the synthetic expansion of limited datasets across distributed nodes which is a critical need in domains where data sharing is tightly regulated due to privacy concerns. Generative models also excel at solving the problem of imbalanced or non-IID (not independently and identically distributed) data that are common in federated learning scenarios. These models can also help balance the dataset by creating synthetic examples in underrepresented classes or distributions.

More importantly, generative modeling can improve privacy of data by creating synthetic datasets statistically similar to the original dataset while not revealing any sensitive information. Possible incorporation of techniques for density estimation and sample generation in [15] in relation to the Rényi Differential Privacy (RDP) and GBDTs framework would support the generation of high fidelity synthetic data while maintaining necessary privacy constraints. Such an integration can result in more diversified and rich data interactions across federated networks. Which inturn will increases the scope and impact of federated learning programs while protecting individual privacy. Although [15] shows federated learning is beneficial in dispersed data contexts, it may not entirely address difficulties in centralised or non-federated systems.

4 Approach

4.1 Design

This research design was constructed using two main components:

- The Adversarial Random Forest Algorithm by David Watson [12] for generative modeling with tree based methods explained in subsection 3.3.
- The Differential Privacy definitions explained in subsection 3.4 and theorems proposed by Cynthia Dwork in [13].

The ARF algorithm which has been implemented in both R and Python, gives us a good foundation to build upon. For this research I used the python package arfpy [16] (Note: There were certain issues related to the alpha parameter in the original forde method of the python package which was dealt with to ensure completeness in the implementation. Details are discussed in the Appendix section A). The Differential Privacy framework provides a way to add controlled noise to the data, ensuring that the synthetic data maintains privacy guarantees. I mainly focus on integrating Laplacian and Exponential mechanisms for numeric and categorical features, respectively, into the arfpy package.

The integration of these mechanisms in the arfpy plays an important role. Raising questions like, Where to add noise ?, How to add noise ? and How much noise to add ?. The arfpy code is well structured and separable thus fewer changes were needed to generate synthetic data which follows differential privacy, Such that instead of the adversary accessing the original dataset with privacy mechanism in place now we present the adversary with a differentially private synthetically generated data which can be accessed by the adversary in a interactive or non interactive way. The point being that when the adversary is only provided with differentially private synthetic data, the need for additional privacy mechanisms on subsequent queries is generally unnecessary. The differential privacy guarantees are embedded within the synthetic data generation process, ensuring that individual privacy is protected regardless of how the synthetic data is analyzed. Adding to this the use of noise in differential privacy makes it difficult for an adversary to trace back the original data points because the noise creates uncertainty and overlaps in the possible outputs. This ensures that individual data contributions remain hidden while allowing the synthetic dataset to retain its overall utility for analysis given it completely follows the differential privacy principles at all critical stages of generation.

- **Identifying Key Queries:** Determining which operations within the arfpy are sensitive and contribute to the synthetic data generation process. The Key Queries were mean, variance, and class probabilities. All the key queries are present in the FORDE method which estimates the density of the data.
- **Calculating Sensitivity:** For each key query, calculate the sensitivity, which is the maximum change in the output due to the addition or removal of a single data record. This is essential for calibrating the amount of noise to be added.

- **Apply Differential Privacy Mechanism:** Use Laplace and Exponential mechanisms to add noise to the outputs of these queries. The amount of noise is determined by the sensitivity and the desired privacy budget.
- **Generate Differential Private Synthetic Data:** Run the modified Differentially Private arfpy(dp arfpy) to produce synthetic data. This data will inherently protect individual privacy due to the noise added during the critical steps of density estimation.

4.2 Theoretical definitions and Theorems

Differential Privacy and its definition as previously discussed in subsection 3.4 along with the below mechanisms which are part of the research from [13] will be used to add noise in the arfpy.

4.2.1 Laplace Mechanism:

To ensure differential privacy for numerical queries, the Laplace mechanism adds noise from a Laplace distribution. The noise scale is determined by the sensitivity of the function, defined as the maximum change in output by adding or removing a single data point, as stated in [13]:

$$K(x) = f(x) + \text{Lap}(\Delta f/\epsilon) \tag{4.1}$$

Here:

- $f(x)$ is the query function.
- Δf is the sensitivity of the function f.
- ϵ is the privacy parameter.
- $\text{Lap}(\Delta f/\epsilon)$ denotes the Laplace noise added to the output.

4.2.2 Exponential Mechanism:

For non-numeric outputs, the exponential mechanism as stated in [13] selects an output based on a utility function, with higher probability given to more desirable outputs. The probability of selecting an output o is proportional to:

$$\Pr[K(D) = o] \propto \exp\left(\frac{\epsilon u(D, o)}{2\Delta u}\right) \tag{4.2}$$

Where:

- $\Pr[K(D) = o]$ is the probability that the mechanism K outputs o when given the dataset D.

- ϵ is the privacy parameter that controls the trade-off between privacy and accuracy.
- $u(D, o)$ is the utility function that measures the quality or utility of the output o given the dataset D.
- Δu is the sensitivity of the utility function, defined as the maximum change in the utility function u when a single element of the dataset D is modified.

4.3 Implementation

The overall goal of integrating differential privacy (DP) into ARFPY is to generate synthetic data that protects individual privacy while preserving data utility.

4.3.1 Enhancing Density Estimation (FORDE) with Differential Privacy

4.3.1.1 Steps for Adding Differential Privacy to Mean for Numeric Features

1. **Sensitivity Calculation for Continuous Variables:**

   Sensitivity of a function (Δf) that calculates the mean (μ) and standard deviation (σ) depends on the the range of data values within each leaf node.

2. **Observed Minimum and Maximum:**

   For each group defined by a tree, node, and variable, calculate the observed minimum ($\min_{\text{observed}}$) and maximum ($\max_{\text{observed}}$) values:

   $$\min_{\text{observed}} = \min(\text{value})$$

   $$\max_{\text{observed}} = \max(\text{value})$$

3. **Sensitivity for Mean:**

   Sensitivity of the mean for each group is given by:

   $$\Delta\mu = \frac{\max_{\text{observed}} - \min_{\text{observed}}}{n}$$

   where n is the number of data points in the group. Here, sensitivity is defined as the maximum change in the mean that results due to the addition or removal of one data point.

4. **Handling Zero Sensitivity:**

   To avoid zero sensitivity, which can occur if all values are identical, we replace zero sensitivity with a small positive value (1×10^{-6}).

5. **Adding Laplace Noise to Mean:**

   The Laplace mechanism adds noise drawn from a Laplace distribution to ensure differential privacy. The noise is calibrated to the sensitivity of the function and the privacy budget (ϵ).

6. **Scale of Laplace Noise:**

   The scale parameter (b) for the Laplace distribution is determined by the sensitivity and privacy budget:

   $$b = \frac{\Delta\mu}{\epsilon}$$

   where ϵ is the privacy budget, controlling the trade-off between privacy and accuracy.

7. **Noisy Mean:**

   Add Laplace noise to the calculated mean:

   $$\mu' = \mu + \text{Lap}\left(\frac{\Delta\mu}{\epsilon}\right)$$

   where $\text{Lap}\left(\frac{\Delta\mu}{\epsilon}\right)$ represents a random draw from the Laplace distribution with scale $\frac{\Delta\mu}{\epsilon}$.

8. **Recalculated Standard Deviation Based on Noisy Mean:**

   After adding noise to the mean, the standard deviation must be recalculated to reflect the new noisy mean (μ').

9. **Noisy Standard Deviation:**

   For each group, recalculate the standard deviation using the noisy mean:

   $$\sigma' = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu')^2}$$

   This recalculation ensures that the variance around the noisy mean is accurately represented.

Theoretical Justification for Differential Privacy Guarantee: By adding Laplace noise proportional to the sensitivity of the mean, this approach ensures that the change in the output due to any single data point is bounded. Thus, providing ϵ-differential privacy. The recalculated standard deviation ensures that the distribution's spread is accurately maintained, preserving data utility. This approach balances privacy and utility by calibrating the noise to the sensitivity of the data. Hence providing a balanced trade-off.

4.3.1.2 Steps for Adding Differential Privacy to Class Probabilities for Categorical features

1. **Calculate Initial Probabilities:**

   For each class within each leaf node, calculate the initial probability based on the count of data points:

   $$\text{prob} = \frac{\text{count\_var\_val}}{\text{count\_var}}$$

   where count_var_val is the count of a specific class and count_var is the total count of all classes within the node.

2. **Use Initial Probabilities as Utility Scores:**

   The initial probabilities are used as the utility scores:

   $$\text{scores} = \text{prob}$$

3. **Sensitivity of the Utility Function:**

   The sensitivity of the utility function in this context is 1. This is because the maximum change in the count (and thus the probability) when a single data point is added or removed is 1.

4. **Apply the Exponential Mechanism:**

   The exponential mechanism is applied by adding noise to the utility scores:

   $$\text{exp\_noise} = \exp\left(\frac{\epsilon \times \text{scores}}{2}\right)$$

5. **Update Probabilities:**

   Update the probabilities with the exponential noise:

   $$\text{prob} = \text{exp\_noise}$$

6. **Normalize Probabilities:**

   Normalize the noisy probabilities within each group to ensure they sum to one:

   $$\text{prob}/ = \sum \text{prob}$$

Theoretical Justification for Differential Privacy Guarantee: By calculating initial probabilities and using them as utility scores in the exponential mechanism, this approach ensures differential privacy for class probabilities within each leaf node. This method balances privacy and utility, allowing for the generation of synthetic data that protects individual data points while preserving the statistical properties of the original dataset.

4.3.2 Synthetic Data Generation (FORGE)

4.3.2.1 Objective: Generate new synthetic data points from the noisy distributions obtained from the density estimation step.

4.3.2.2 Steps:

- **Sample Data:** For continuous features, sample from the truncated normal distribution using noisy μ' and σ'. For categorical features, sample according to the noisy class probabilities.
- **Combine Features:** Combine sampled features to create synthetic data points that preserve the statistical properties of the original dataset while ensuring privacy.

Note: In the end some post processing is required to ensure synthetic data looks exactly like the original data this is the case for both the Original arfpy (og arfpy) and (Differentially Private arfpy) dp arfpy.

4.4 Alternate and Failed Approaches

Here, I present and discuss the alternate approaches that could have been used and also the failed approaches that were used to enhance generative tree based modeling with differential privacy. Each approach had distinct strategies and implementations.

4.4.1 Federated Boosted Differentially Private Generative Trees

The goal was to implement generative modeling into [15] to for the generation of synthetic data that satisfies differential privacy guarantees.

- **Density Estimation:** After the trees was trained, densities were estimated by traversing the trees and gathering the statistics (mean, standard deviation) of the data points in each leaf node. To these statistics, I added noise to ensure differential privacy.
- **Gradient, Splits and Hessian-Based Generation:** Using noisy gradients and Hessians from the training process to guide the generation of synthetic data. The Synthetic data points were generated by randomly choosing paths in the trees and sampling data points within the defined splits. Noise was added to the decision boundaries to ensure privacy.

Overall, While integrating generative modeling within the Federated Boosted DP Trees framework is theoretically appealing, but in practice, it suffers greatly in the effort to balance privacy versus data utility, capture complex data relationships, and manage computational overhead. Further research is needed to integrate effective generative modeling into the Federated Boosted Decision Trees with differential privacy framework.

4.4.2 Differentially Private Generative Trees and Forests or Adversarial Random Forest

The goal was to enhance existing generative modeling algorithms given in [10], [11], and [12] with differential privacy to generate high-quality synthetic data that preserves the privacy of individuals in the original dataset.

When deciding to implement differential privacy (DP) in ARFPY over the generative forest, several key factors were considered. ARFPY has been well modularized, and has more distinct task separation, including density estimation and data generation phases, which helps a lot to integrate DP. Given the structure of the model it is, therefore, much easier to implement and have a low cost privacy assurance as compared to extensive code refactoring.

On the other hand, the approach with generative forests is not the favorite one but potentially plausible for elaborate multi-dimensional models of data. Its design embodies adversarial loops and tree based modeling able to handle intricate data interactions, hence being adaptive to diverse datasets. Therein lies the flexibility to adjust and fine-tune privacy parameters, which are very important for the discipline of privacy preserving data synthesis. While challenges will remain, such as balancing cumulative noise with consistent privacy preservation, the generative forest should be able to effectively calibrate with sufficient testing in order to preserve accuracy and privacy. That way, it makes it highly suitable for advanced differential privacy applications and a robust framework for future explorations in the field.

4.5 Experimental Design

4.5.1 Data Acquisition

The datasets are preprocessed, to be specific One Hot Encoding for categorical variables is done for evaluation and not for training the arfpy models because that is already handled internally in arfpy. Also I have manually ensured equal decimal digit across all numerical features and correct data type for all features as well.

- **Health Insurance Dataset for US** [17]:
  - Contains insurance charges for 1338 people along with their personal data like Age, Body Mass Index (BMI), Sex, Smoker, Number of children, Region.
  - It is a synthetic dataset based on US census data.
- **UCI Adult Dataset** [18]:
  - The UCI Machine Learning Repository's Adult Dataset contains 48,842 instances with both continuous and categorical features, taken from the 1994 United States Census..
  - This dataset includes sensitive and specific information about individuals such as age, work class, education, marital status, occupation, race, gender, native

country, and income, which is categorized as either greater than or less than $50,000 per year.
  - it is important to note that it is composed of real, anonymized data and therefore is used in compliance with privacy regulations as given by the source.

- **Healthcare Dataset** [19]:
  - Contains 55,500 rows of sensitive and specific information about the patients including their age, blood type, medical condition, insurance provider, room number, billing amount, and others.
  - It is entirely synthetic implying it does not contain any real patient information or violates any privacy regulations.

4.5.2 Metrics and Plots for Evaluation

Overview of the technical plots and metrics used for evaluation of original dataset, og arfpy and dp arfpy generated datasets along with their importance.

- **Density Plot:** A graphical representation of the data distribution of the numeric variables. It can be interpreted as a smoothed version of a histogram. This will help us compare the distribution of the features visually.
- **Principal Component Analysis (PCA) Plot:** PCA is a dimensionality reduction technique employed to find out the direction in which the data varies the most. This will help compare the variability and importance across the datasets showing whether the utility is preserved. The first few principal components retain most of the variation present in the original dataset. Thus doing the PCA of top 2 components.
- **Correlation:** It measures the extent to which two variables are linearly related (strength and direction). This will help compare the change in relationship between the datasets.
- **Wasserstein Distance:** It provides a way of measuring the difference between two probability distributions. The approach is expressive in measuring the differences in the distributions when considering how the synthetic data was manipulated in relation to the original one.
- **Chi-Square Test:** An exclusive categorical variable test. To identify if there is an association between the categorical variables. It checks for goodness of fit between the observed data and the expected distribution or if the differences are present in the observed and the expected frequencies between original and synthetic datasets.
- **Root Mean Square Error (RMSE) and Mean Absolute Error (MAE):** It is a measure that checks if predictions are accurate. The RMSE calculates the square root of the average of squared differences between predicted and actual

values and takes more weight on larger errors, being sensitive to outliers. Averaging absolute differences, MAE gives the average error. Since it is not so influenced by outliers, this provides a rather direct measure of average error. RMSE is sensitive to error sizes, whereas MAE gives a more reasonable and overall good accuracy of the model. It also gives us a measurable estimate of differences in distributions between the original and synthetic datasets.

- **Classification Accuracy:** It is a measure of the model's accuracy, which has been trained with original data and tested with synthetic data. It helps in understanding whether the utility is preserved.

- **Reconstruction Attack:** An attack through which the dissimilarity between an original dataset and a synthetic dataset is measured by how well the synthetic data can reconstruct the original data. The k-NN algorithm evaluates how near synthetic points are to the original points. First it initializes the k-NN model to look for the three closest neighbors in the synthetic dataset for each point in the original dataset. It then calculates distance between each of the original data points and their nearest synthetic neighbors, averaging those distances together in order to produce a measure of how well the synthetic data reconstructs the original data. The mean distance is returned by the function, meaning the smaller the value, the better the reconstruction attack is in simulating the original data with synthetic data. This is a valuable measure used in assessing the privacy of synthetically generated data.

5 Results and Evaluation

. In this section, I present the results of my evaluation of the Differential Privacy (DP) enhanced ARFPY model. I address my research questions through quantitative, qualitative, and visual analyses. Detailed comparisons among the real data, data generated with the Original ARF, and data generated with the new DP-ARF are provided in order to evaluate the ARF models. This comparison involves a number of metrics that include density distributions, overlaps in PCA, Wasserstein distance, correlation matrices, and classification accuracy. Moreover, the effectiveness of privacy preservation is incorporated in the analysis with Reconstruction Attack, all of which is explained in subsubsection 4.5.2.

5.1 Health Insurance Dataset

5.1.1 Epsilon = 0.5

age	sex	bmi	children	smoker	region	charges
19	female	27.90	0	yes	southwest	16884.92
18	male	33.77	1	no	southeast	1725.55
28	male	33.00	3	no	southeast	4449.46
33	male	22.71	0	no	northwest	21984.47
32	male	28.88	0	no	northwest	3866.86

Table 1: Original Data for Insurance $\epsilon = 0.5$

age	sex	bmi	children	smoker	region	charges
76	female	31.29	0	yes	southwest	64002.06
62	female	22.88	0	no	northeast	10853.21
58	male	34.43	0	no	northwest	11382.47
55	female	20.14	0	no	southwest	11566.45
28	male	32.92	1	no	southwest	14846.36

Table 2: OG ARF Generated Data for Insurance $\epsilon = 0.5$

age	sex	bmi	children	smoker	region	charges
25	female	41.02	1	yes	southeast	41584.19
74	female	30.50	0	no	southwest	10590.04
44	male	32.58	3	no	northwest	7950.42
46	female	22.39	1	yes	southwest	19380.03
55	male	24.46	0	yes	southwest	12792.37

Table 3: DP-ARF Generated Data for Insurance $\epsilon = 0.5$

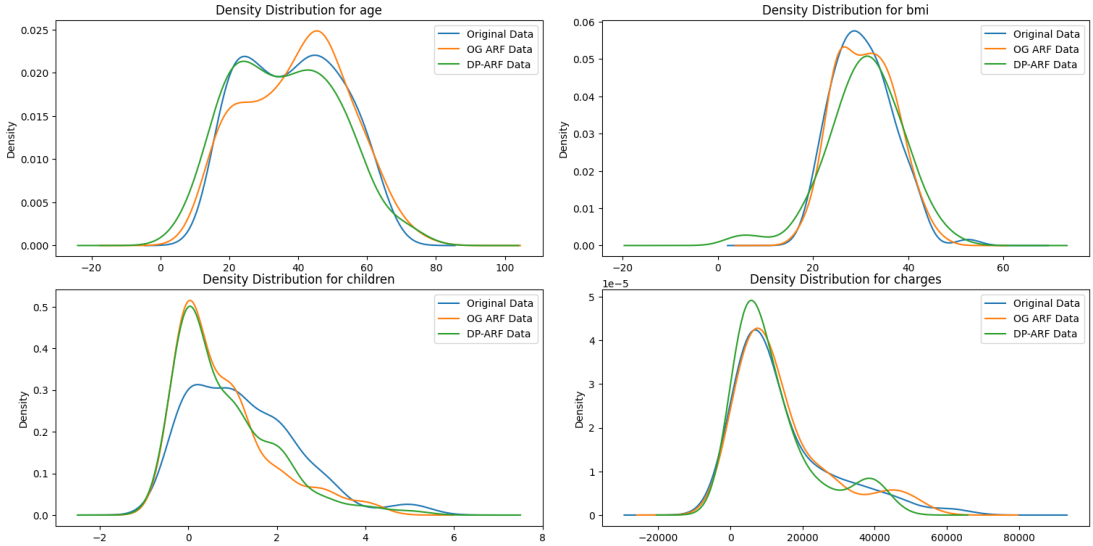

Figure 1: Density distribution for Insurance $\epsilon = 0.5$

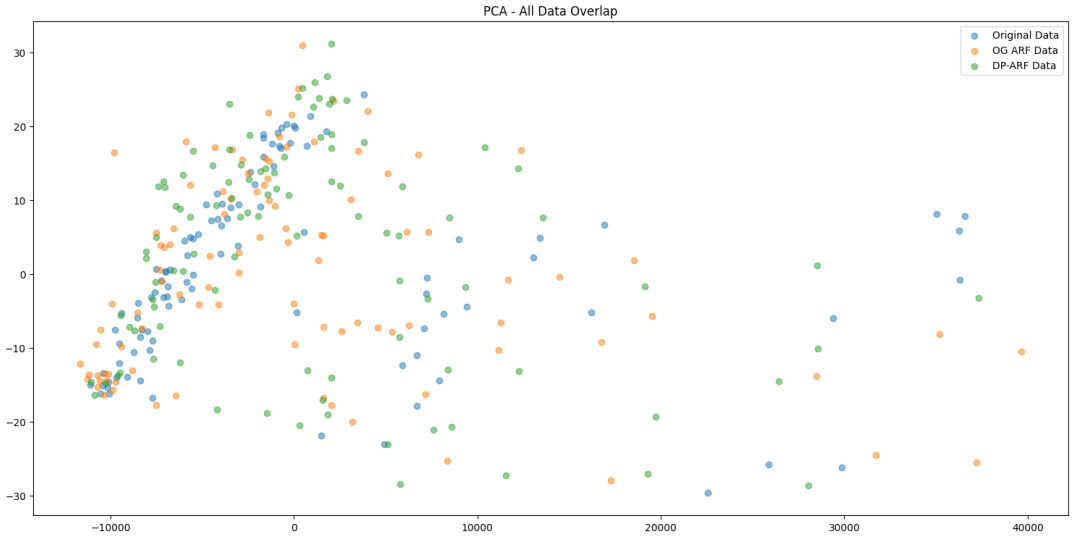

Figure 2: PCA analysis for Insurance $\epsilon = 0.5$

Features	Original Data	OG ARF Data	DP ARF Data
Age vs BMI	0.1093	0.1193	0.1018
Age vs Children	0.0425	0.0171	0.0832
Age vs Charges	0.2990	0.3117	0.2404
BMI vs Children	0.0128	-0.0393	0.0064
BMI vs Charges	0.1983	0.1817	0.1843
Children vs Charges	0.0680	0.0527	0.0437

Table 4: Correlation Matrices for Insurance $\epsilon = 0.5$

Feature	OG ARF	DP ARF
Age	0.7108	0.7519
BMI	0.1709	0.7577
Children	0.2324	0.2840
Charges	737.2693	1173.4129

Table 5: Wasserstein Distance for Insurance $\epsilon = 0.5$

Feature	OG ARF	DP ARF
Sex	0.0120	0.0478
Smoker	0.1132	1.9177
Region	1.4885	4.0125

Table 6: Chi-Square Test for Insurance $\epsilon = 0.5$

Feature	OG ARF	DP ARF
Age	19.8058	20.5487
BMI	8.7060	9.4203
Children	1.6591	1.6656
Charges	17254.7497	17888.7417

Table 7: RMSE for Insurance $\epsilon = 0.5$

Feature	OG ARF	DP ARF
Age	16.2324	16.7773
BMI	6.7161	7.3835
Children	1.2294	1.2407
Charges	12686.0164	12687.9253

Table 8: MAE for Insurance $\epsilon = 0.5$

ARF	Value
OG	0.9940
DP	0.9821

Table 9: Classification Accuracy for Insurance $\epsilon = 0.5$

ARF	Value
OG	1.1491
DP	1.1665

Table 10: Reconstruction Attack Error for Insurance $\epsilon = 0.5$

5.1.2 Epsilon = 0.1

age	sex	bmi	children	smoker	region	charges
19	female	27.90	0	yes	southwest	16884.92
18	male	33.77	1	no	southeast	1725.55
28	male	33.00	3	no	southeast	4449.46
33	male	22.71	0	no	northwest	21984.47
32	male	28.88	0	no	northwest	3866.86

Table 11: Original Data for Insurance $\epsilon = 0.1$

age	sex	bmi	children	smoker	region	charges
28	female	34.72	1	no	southeast	3525.54
19	male	15.38	0	no	northeast	1552.01
48	female	26.94	0	no	northwest	13151.98
19	female	29.46	0	no	northwest	2164.71
52	female	38.00	1	no	northwest	39109.23

Table 12: OG ARF Generated Data for Insurance $\epsilon = 0.1$

age	sex	bmi	children	smoker	region	charges
31	male	33.68	0	no	southwest	17180.43
152	female	40.28	0	no	northwest	36628.09
47	female	33.47	4	no	southwest	8072.56
-7	male	47.51	2	yes	southwest	40062.53
33	female	34.41	2	no	southeast	8511.86

Table 13: DP-ARF Generated Data for Insurance $\epsilon = 0.1$

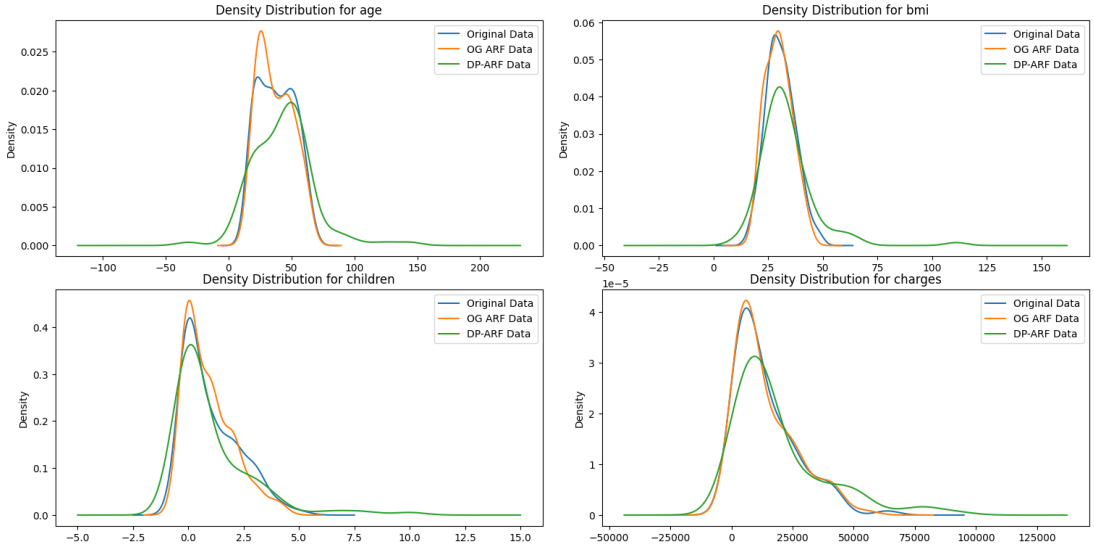

Figure 3: Density distribution for Insurance $\epsilon = 0.1$

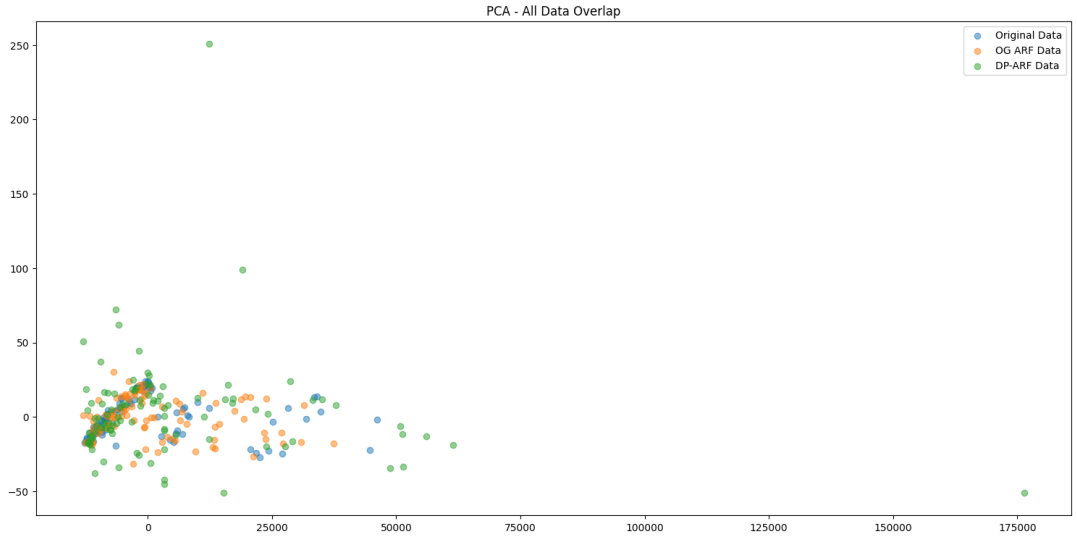

Figure 4: PCA analysis for Insurance $\epsilon = 0.1$

Features	Original Data	OG ARF Data	DP ARF Data
Age vs BMI	0.1093	0.1075	-0.0135
Age vs Children	0.0425	0.0516	0.0220
Age vs Charges	0.2990	0.2752	0.1215
BMI vs Children	0.0128	0.0765	-0.0110
BMI vs Charges	0.1983	0.1935	0.0784
Children vs Charges	0.0680	0.0898	0.0646

Table 14: Correlation Matrices for Insurance $\epsilon = 0.1$

Feature	OG ARF	DP ARF
Age	0.7960	5.4372
BMI	0.1860	2.6262
Children	0.1771	0.5628
Charges	444.6989	3934.5369

Table 15: Wasserstein Distance for Insurance $\epsilon = 0.1$

Feature	OG ARF	DP ARF
Sex	0.0030	1.7231
Smoker	1.9438	0.0410
Region	2.0303	2.1192

Table 16: Chi-Square Test for Insurance $\epsilon = 0.1$

Feature	OG ARF	DP ARF
Age	19.6901	34.9413
BMI	8.6822	13.4745
Children	1.6532	3.3059
Charges	17012.1896	25427.7295

Table 17: RMSE for Insurance $\epsilon = 0.1$

Feature	OG ARF	DP ARF
Age	16.2055	20.8318
BMI	7.1869	9.2910
Children	1.2773	1.6958
Charges	12064.6710	15174.2791

Table 18: MAE for Insurance $\epsilon = 0.1$

ARF	Value
OG	0.9948
DP	0.9454

Table 19: Classification Accuracy for Insurance $\epsilon = 0.1$

ARF	Value
OG	1.1301
DP	1.2600

Table 20: Reconstruction Attack Error for Insurance $\epsilon = 0.1$

5.1.3 Epsilon = 1.0

age	sex	bmi	children	smoker	region	charges
19	female	27.90	0	yes	southwest	16884.92
18	male	33.77	1	no	southeast	1725.55
28	male	33.00	3	no	southeast	4449.46
33	male	22.71	0	no	northwest	21984.47
32	male	28.88	0	no	northwest	3866.86

Table 21: Original Data for Insurance $\epsilon = 1$

age	sex	bmi	children	smoker	region	charges
51	female	40.59	1	no	southeast	12974.91
33	female	21.02	0	no	southwest	16722.54
27	female	27.85	0	yes	southwest	47654.03
53	female	33.83	0	no	southwest	11571.54
20	male	31.50	0	no	southwest	1150.92

Table 22: OG ARF Generated Data for Insurance $\epsilon = 1$

age	sex	bmi	children	smoker	region	charges
22	female	24.30	0	no	southeast	2654.25
26	male	27.27	1	no	southwest	3103.97
32	male	33.37	0	no	northwest	13654.07
49	male	31.00	0	no	southeast	20908.33
49	male	35.69	0	no	southeast	15504.31

Table 23: DP-ARF Generated Data for Insurance $\epsilon = 1$

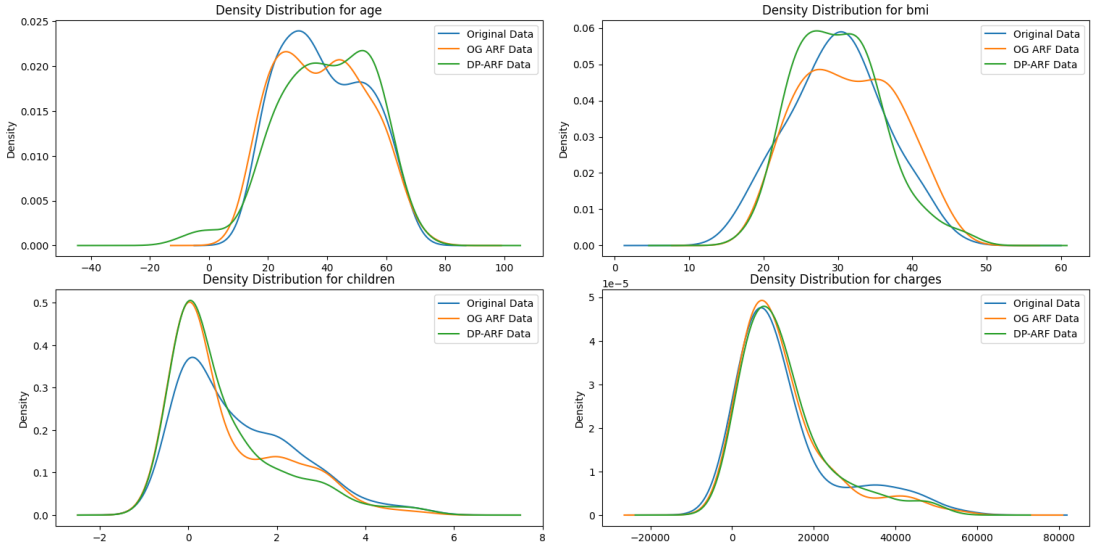

Figure 5: Density distribution for Insurance $\epsilon = 1$

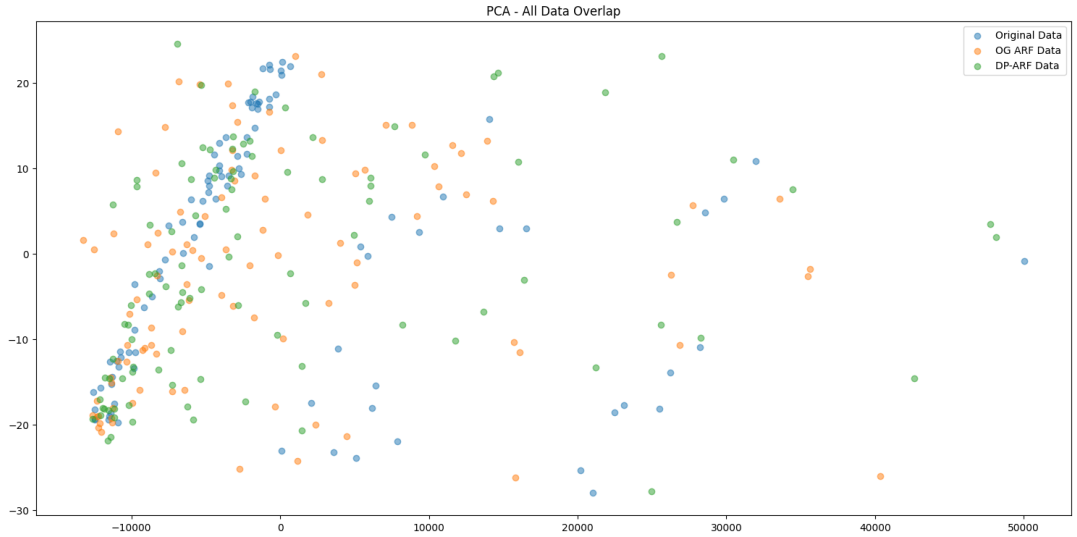

Figure 6: PCA analysis for Insurance $\epsilon = 1$

Features	Original Data	OG ARF Data	DP ARF Data
Age vs BMI	0.1093	0.1136	0.1063
Age vs Children	0.0425	0.0671	0.0578
Age vs Charges	0.2990	0.2891	0.3368
BMI vs Children	0.0128	0.0323	-0.0047
BMI vs Charges	0.1983	0.1745	0.1376
Children vs Charges	0.0680	0.0672	0.0876

Table 24: Correlation Matrices for Insurance $\epsilon = 1$

Feature	OG ARF	DP ARF
Age	0.9641	1.1121
BMI	0.2340	0.3356
Children	0.2556	0.2392
Charges	622.5091	757.9146

Table 25: Wasserstein Distance for Insurance $\epsilon = 1$

Feature	OG ARF	DP ARF
Sex	0.0120	1.7231
Smoker	0.0742	5.4413
Region	2.4009	1.5549

Table 26: Chi-Square Test for Insurance $\epsilon = 1$

Feature	OG ARF	DP ARF
Age	19.6970	20.3337
BMI	8.7164	8.9149
Children	1.6316	1.6430
Charges	16630.1894	17068.2742

Table 27: RMSE for Insurance $\epsilon = 1$

Feature	OG ARF	DP ARF
Age	16.1913	17.0942
BMI	7.0242	7.1800
Children	1.2407	1.2287
Charges	12397.7754	12610.3587

Table 28: MAE for Insurance $\epsilon = 1$

ARF	Value
OG	0.9955
DP	0.9851

Table 29: Classification Accuracy for Insurance $\epsilon = 1$

ARF	Mean Distance
OG	1.1295
DP	1.1449

Table 30: Reconstruction Attack Error for Insurance $\epsilon = 1$

Comparative density plots for characteristics like age, BMI, children, and charges at different DP settings (epsilon values 0.1, 0.5, and 1) are shown in Figure 1, Figure 3, and Figure 5 respectively. Overall, The plots demonstrates that the DP-ARF data generally follows the distribution of the original and non-private ARF (OG ARF) data. However, as expected with higher privacy (lower epsilon values) we get more distortion and deviation.

The graphical representation of PCA in Figure 2, Figure 4, and Figure 6 showed a pronounced overlap, implying that the principal components of synthetic data preserve the relationships and capture the important variations of variables. Although there are some increased noisy points in variance due to noise addition, especially noticeable at lower epsilon values.

Evaluation metrics such as Wasserstein distance, chi-square tests, RMSE, and MAE shows the differences between the distributions of the original and synthetic datasets. Lower values of epsilon showed increase in differences and error metrics, again its due to the trade-off between privacy and data utility.

Further validations on the utility of the synthetic data were made through classification tasks. Models performance was a little worse for the DP-ARF models compared to OG ARF in Table 9 and Table 29, but the difference is large especially at stricter privacy levels as shown in Table 19. This is because of noise negatively impacting predictive quality.

Finally, the reconstruction attack scenario tested how well an adversary could reconstruct original entries from synthetic data. The results in Table 10, Table 20 and Table 30 show an increase in mean distances. thus indicating effective increase in privacy levels.

Overall, the results for the health insurance dataset shows the effectiveness of DP-ARF in producing usable synthetic data with good privacy guarantees, The best trade-off between utility and privacy was found at epsilon 0.5.

5.2 Healthcare Dataset

5.2.1 Epsilon = 0.5

Name	Age	Gender	Blood Type	Medical Condition
Bobby Jackson	30	Male	B-	Cancer
Leslie Terry	62	Male	A+	Obesity
Danny Smith	76	Female	A-	Obesity
Andrew Watts	28	Female	O+	Diabetes
Adrienne Bell	43	Female	AB+	Cancer

Table 31: Original Healthcare Data Part 1 $\epsilon = 0.5$

Billing Amount	Room Number	Medication	Test Results
18856.28	328	Paracetamol	Normal
33643.33	265	Ibuprofen	Inconclusive
27955.10	205	Aspirin	Normal
37909.78	450	Ibuprofen	Abnormal
14238.32	458	Penicillin	Abnormal

Table 32: Original Healthcare Data Part 2 $\epsilon = 0.5$

Name	Age	Gender	Blood Type	Medical Condition
Sarah Garcia	46	Female	O-	Diabetes
Mario Moreno	44	Male	O+	Diabetes
David Sweeney	52	Male	B+	Arthritis
Donald Morrison	65	Male	B+	Diabetes
Daniel Blankenship	81	Male	A+	Asthma

Table 33: OG ARF Generated Healthcare Data Part 1 $\epsilon = 0.5$

Billing Amount	Room Number	Medication	Test Results
29269.15	230	Paracetamol	Inconclusive
36451.28	225	Penicillin	Inconclusive
13229.58	196	Lipitor	Normal
2490.91	274	Aspirin	Inconclusive
39561.80	200	Paracetamol	Inconclusive

Table 34: OG ARF Generated Healthcare Data Part 2 $\epsilon = 0.5$

Name	Age	Gender	Blood Type	Medical Condition
Carolyn Rowe	76	Male	B-	Diabetes
Kevin Hendrix	54	Female	O+	Asthma
Ryan Taylor	64	Female	A-	Cancer
Ashley Walter	22	Female	AB-	Obesity
John Thompson	29	Male	O-	Arthritis

Table 35: DP ARF Generated Healthcare Data Part 1 $\epsilon = 0.5$

Billing Amount	Room Number	Medication	Test Results
50929.46	303	Aspirin	Abnormal
9633.21	247	Paracetamol	Inconclusive
21886.56	186	Paracetamol	Abnormal
8988.97	464	Aspirin	Normal
14689.24	192	Ibuprofen	Normal

Table 36: DP ARF Generated Healthcare Data Part 2 $\epsilon = 0.5$

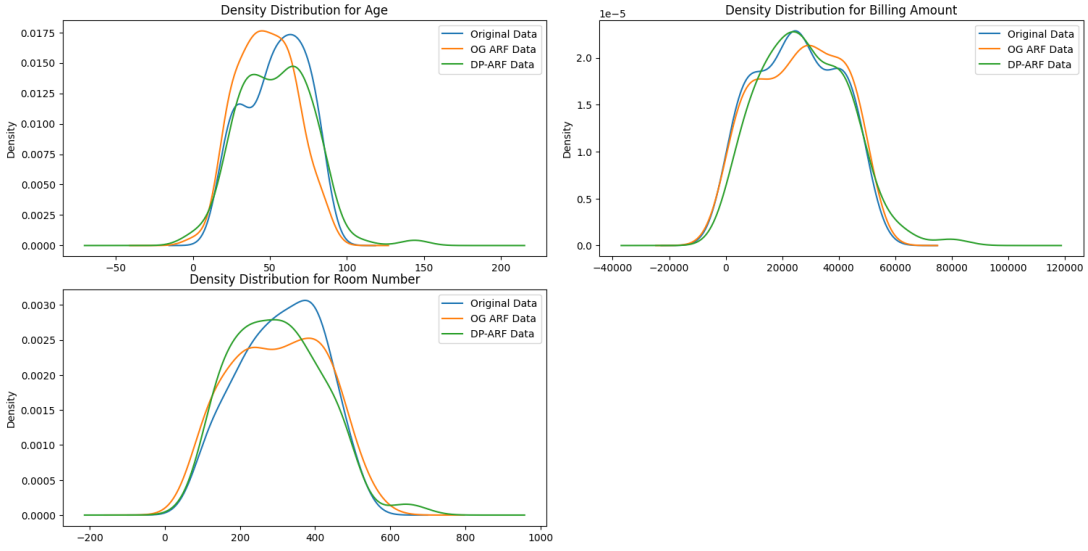

Figure 7: Density distribution for Healthcare $\epsilon = 0.5$

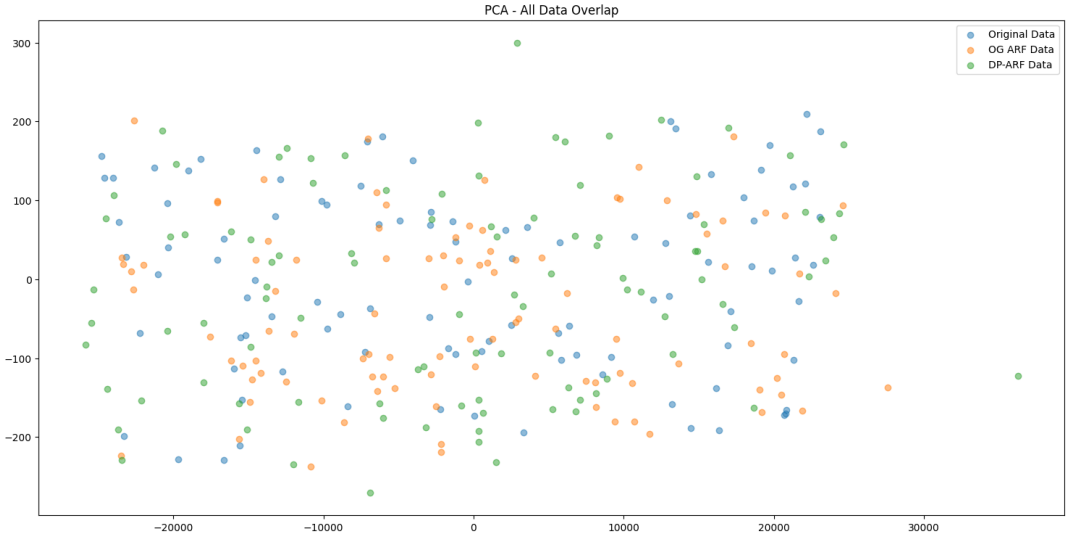

Figure 8: PCA analysis for Healthcare $\epsilon = 0.5$

Features	Original Data	OG ARF Data	DP ARF Data
Age vs BMI	0.1093	0.1193	0.1018
Age vs Children	0.0425	0.0171	0.0832
Age vs Charges	0.2990	0.3117	0.2404
BMI vs Children	0.0128	-0.0393	0.0064
BMI vs Charges	0.1983	0.1817	0.1843
Children vs Charges	0.0680	0.0527	0.0437

Table 37: Correlation Matrices for Healthcare $\epsilon = 0.5$

Feature	OG ARF	DP ARF
Age	1.2378	1.4585
Billing Amount	1050.3541	1251.9999
Room Number	5.7045	7.1170

Table 38: Wasserstein Distance for Healthcare $\epsilon = 0.5$

Feature	OG ARF	DP ARF
Sex	1.3330	2.0829
Smoker	4.1318	6.8886
Region	9.6889	5.0011

Table 39: Chi-Square Test for Healthcare $\epsilon = 0.5$

Feature	OG ARF	DP ARF
Age	27.5775	29.2997
Billing Amount	19591.1242	20457.3572
Room Number	161.5081	168.8140

Table 40: RMSE for Healthcare $\epsilon = 0.5$

Feature	OG ARF	DP ARF
Age	22.4694	23.3947
Billing Amount	16048.8418	16459.2068
Room Number	131.3888	135.4193

Table 41: MAE for Healthcare $\epsilon = 0.5$

ARF	Value
OG	0.8256
DP	0.8242

Table 42: Classification Accuracy for Healthcare $\epsilon = 0.5$

ARF	Value
OG	0.4409
DP	0.4673

Table 43: Reconstruction Attack Error for Healthcare $\epsilon = 0.5$

5.2.2 Epsilon = 0.1

Name	Age	Gender	Blood Type	Medical Condition
Bobby Jackson	30	Male	B-	Cancer
Leslie Terry	62	Male	A+	Obesity
Danny Smith	76	Female	A-	Obesity
Andrew Watts	28	Female	O+	Diabetes
Adrienne Bell	43	Female	AB+	Cancer

Table 44: Original Healthcare Data Part 1 $\epsilon = 0.1$

Billing Amount	Room Number	Medication	Test Results
18856.28	328	Paracetamol	Normal
33643.33	265	Ibuprofen	Inconclusive
27955.10	205	Aspirin	Normal
37909.78	450	Ibuprofen	Abnormal
14238.32	458	Penicillin	Abnormal

Table 45: Original Healthcare Data Part 2 $\epsilon = 0.1$

Name	Age	Gender	Blood Type	Medical Condition
Amanda Patel	51	Male	A+	Obesity
Karen Thomas	65	Female	B-	Diabetes
Faith Mathis	83	Female	O+	Diabetes
Jeffery Johnson	27	Male	B-	Asthma
Keith Wiley	58	Female	A+	Arthritis

Table 46: OG ARF Generated Healthcare Data Part 1 $\epsilon = 0.1$

Billing Amount	Room Number	Medication	Test Results
13133.91	419	Ibuprofen	Abnormal
36187.58	407	Paracetamol	Abnormal
37165.47	380	Aspirin	Inconclusive
28384.04	149	Penicillin	Inconclusive
19314.20	496	Paracetamol	Normal

Table 47: OG ARF Generated Healthcare Data Part 2 $\epsilon = 0.1$

Name	Age	Gender	Blood Type	Medical Condition
Holly Dunn	73	Female	B-	Obesity
Keith Wheeler	67	Female	AB-	Asthma
Amanda Richmond	21	Male	AB-	Arthritis
Marissa Guzman	64	Female	AB+	Diabetes
Matthew Cooley	27	Male	AB-	Asthma

Table 48: DP ARF Generated Healthcare Data Part 1 $\epsilon = 0.1$

Billing Amount	Room Number	Medication	Test Results
14524.34	77	Penicillin	Abnormal
18283.76	1165	Ibuprofen	Inconclusive
15898.70	351	Lipitor	Abnormal
27276.49	147	Lipitor	Inconclusive
46902.64	485	Paracetamol	Inconclusive

Table 49: DP ARF Generated Healthcare Data Part 2 $\epsilon = 0.1$

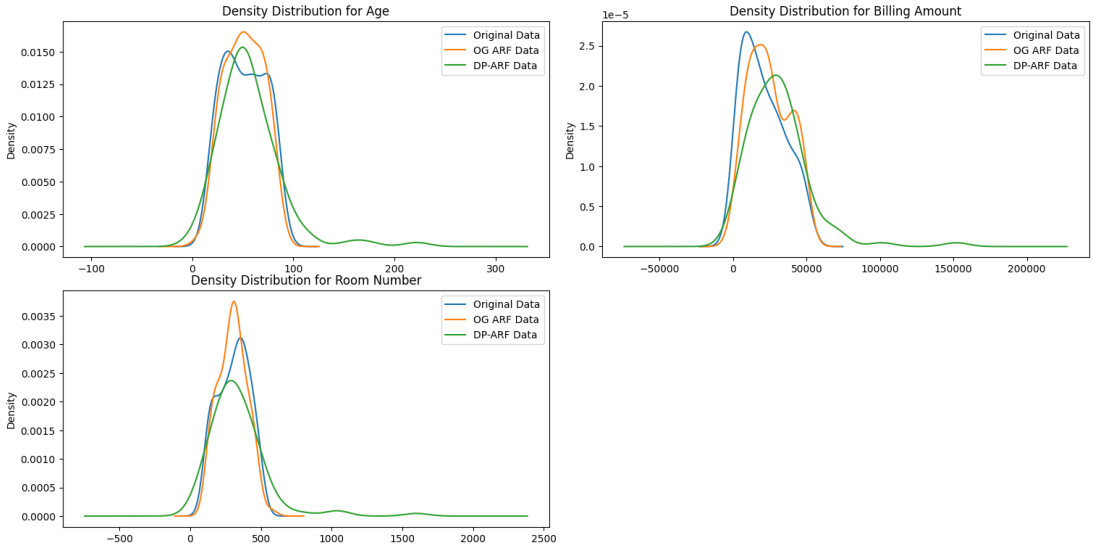

Figure 9: Density distribution for Healthcare $\epsilon = 0.1$

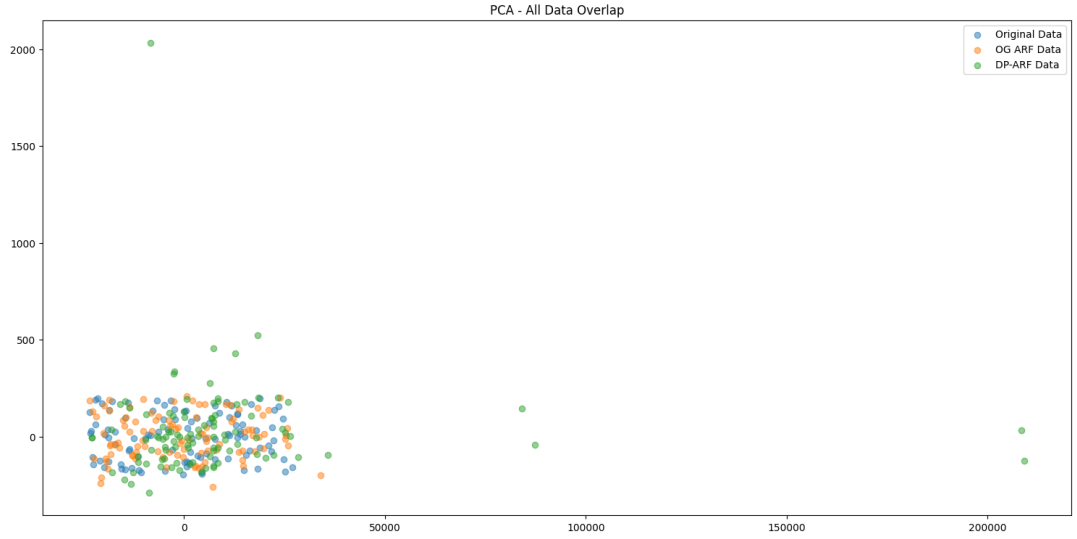

Figure 10: PCA analysis for Healthcare $\epsilon = 0.1$

Features	Original Data	OG ARF Data	DP ARF Data
Age vs Billing Amount	-0.0038	-0.0127	-0.0030
Age vs Room Number	-0.0007	0.0019	0.0059
Billing Amount vs Room Number	-0.0029	0.0004	0.0158

Table 50: Correlation Matrices for Healthcare $\epsilon = 0.1$

Feature	OG ARF	DP ARF
Age	1.1992	8.2768
Billing Amount	1097.5450	5858.5344
Room Number	7.3377	46.9655

Table 51: Wasserstein Distance for Healthcare $\epsilon = 0.1$

Feature	OG ARF	DP ARF
Gender	0.0348	0.1395
Blood Type	5.3213	5.3272
Medical Condition	4.7474	5.7709
Admission Type	5.1291	11.7552
Medication	4.4191	1.9580
Test Results	4.4448	1.6439

Table 52: Chi-Square Tests for Healthcare $\epsilon = 0.1$

Feature	OG ARF	DP ARF
Age	27.5755	49.5280
Billing Amount	19592.9018	32565.8476
Room Number	160.7341	280.8843

Table 53: RMSE for Healthcare $\epsilon = 0.1$

Feature	OG ARF	DP ARF
Age	22.4236	29.7468
Billing Amount	15965.1953	20214.3915
Room Number	130.8158	171.6214

Table 54: MAE for Healthcare $\epsilon = 0.1$

ARF	Value
OG	0.8269
DP	0.8262

Table 55: Classification Accuracy for Healthcare $\epsilon = 0.1$

ARF	Value
OG	0.4476
DP	0.5937

Table 56: Reconstruction Attack Error for Healthcare $\epsilon = 0.1$

Similar to the Insurance Dataset in subsection 5.1 we can see some of the realistic samples generated data by DP-ARF shown in Table 35, Table 36 and OG ARF shown in Table 33, Table 34. And after plotting density distributions for epsilon values of 0.1 Figure 9, 0.5 Figure 7 we can see how synthetic data aligns with the original dataset's distributions for key attributes like Age, Room Number, and Billing Amount. DP-ARF data trends very similarly to that of the original and OG ARF data in most cases and is often with remarkable fidelity even under different privacy settings. Noticeable distortions were introduced by lower epsilon values as the noise level increased, more evident at epsilon = 0.1, which, despite the noise, was determined to be the best trade-off between privacy and utility.

Principal component analysis in Figure 10 and Figure 10 demonstrates a good overlap in data points for the original, OG ARF, and DP-ARF datasets, thus showing that synthetic data holds essential relationships among variables.

The computed statistical metrics, which include Wasserstein distance, chi-square tests, Root Mean Square Error, and Mean Absolute Error, shown in their respective tables demonstrate how well synthetic data mimic the original distributions.

Results of classification task in Table 55 and Table 42 indicates that models evaluated using DP-ARF data for the healthcare dataset are as effective as those trained with OG ARF data even though there is added noise it does not deteriorate the predictive power of the models too much. Thus, retaining good accuracy values in compensation for significant increase in privacy.

Reconstruction attacks demonstrate a higher mean distance error with more stringent privacy settings shown in Table 56 with lower epsilon value of 0.1 illustrating the overall protection against possible data breaches.

In summary, the results on the healthcare dataset underline that DP-ARF can produce effective synthetic datasets that retain most of the utility while guaranteeing significant privacy levels. For the healthcare dataset the chosen epsilon value of 0.1 reflects an optimal trade-off that will maximize privacy while keeping reasonable utility. Additional results if necessary for epsilon value of 1 are shown in Appendix section A.

5.3 Adult Dataset

5.3.1 Epsilon = 0.5

age	**workclass**	**fnlwgt**
39	State-gov	77516
50	Self-emp-not-inc	83311
38	Private	215646
53	Private	234721
28	Private	338409

Table 57: Original Data Sample Part 1 for Adult $\epsilon = 0.5$

education	**hours_per_week**	**native_country**	**income**
Bachelors	40	United-States	$\leq$ 50K
Bachelors	13	United-States	$\leq$ 50K
HS-grad	40	United-States	$\leq$ 50K
11th	40	United-States	$\leq$ 50K
Bachelors	40	Cuba	$\leq$ 50K

Table 58: Original Data Sample Part 2 for Adult $\epsilon = 0.5$

age	workclass	fnlwgt
44	Federal-gov	206408
16	Private	278898
53	Private	372365
46	Private	184376
37	Private	330176

Table 59: OG ARF Generated Adult Data Part 1 $\epsilon = 0.5$

education	hours_per_week	native_country	income
Masters	42	United-States	$>$ 50K
HS-grad	18	United-States	$\leq$ 50K
PhD	48	United-States	$>$ 50K
Bachelors	37	United-States	$\leq$ 50K
HS-grad	64	El-Salvador	$\leq$ 50K

Table 60: OG ARF Generated Adult Data Part 2 $\epsilon = 0.5$

age	workclass	fnlwgt
26	Private	248347
33	Self-emp-inc	31230
53	Private	340454
38	Private	29693
109	Private	122824

Table 61: DP ARF Generated Adult Data Part 1 $\epsilon = 0.5$

education	hours_per_week	native_country	income
Assoc	48	Mexico	$\leq$ 50K
Masters	42	England	$\leq$ 50K
Bachelors	20	Germany	$>$ 50K
Some-college	30	Philippines	$\leq$ 50K
PhD	42	United-States	$>$ 50K

Table 62: DP ARF Generated Adult Data Part 2 $\epsilon = 0.5$

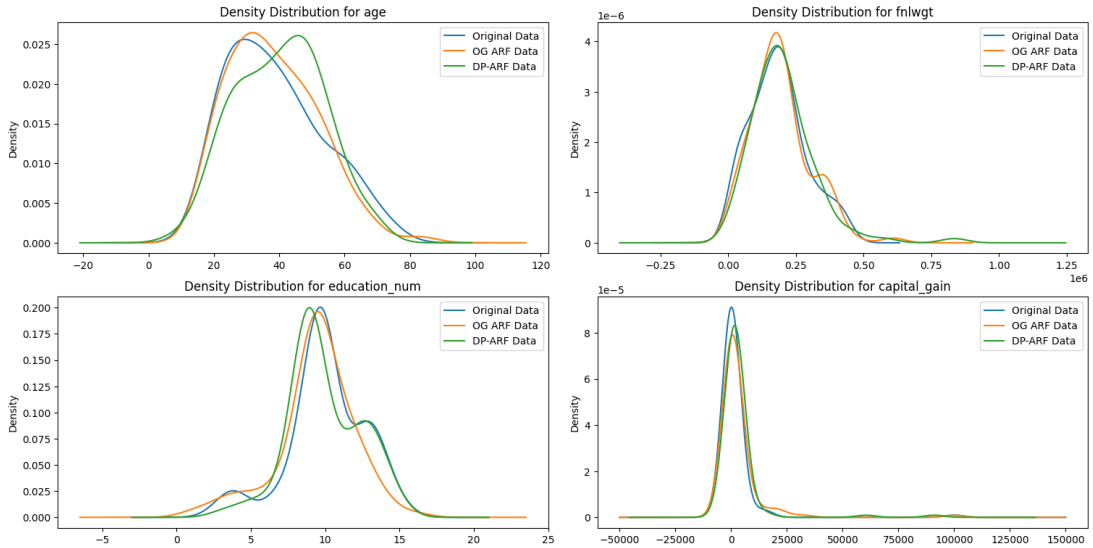

Figure 11: Density distribution for Adult Part 1 $\epsilon = 0.5$

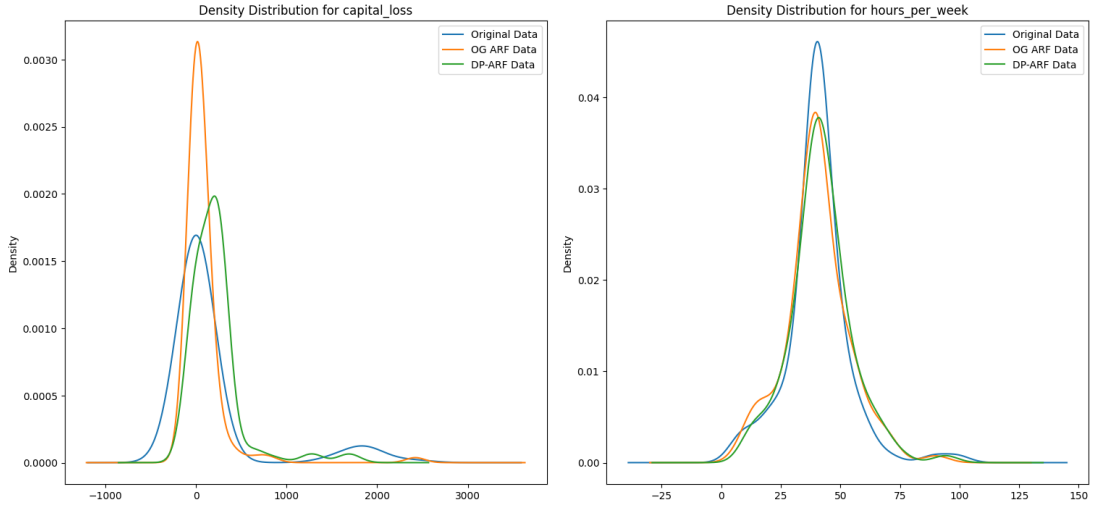

Figure 12: Density distribution for Adult Part 2 $\epsilon = 0.5$

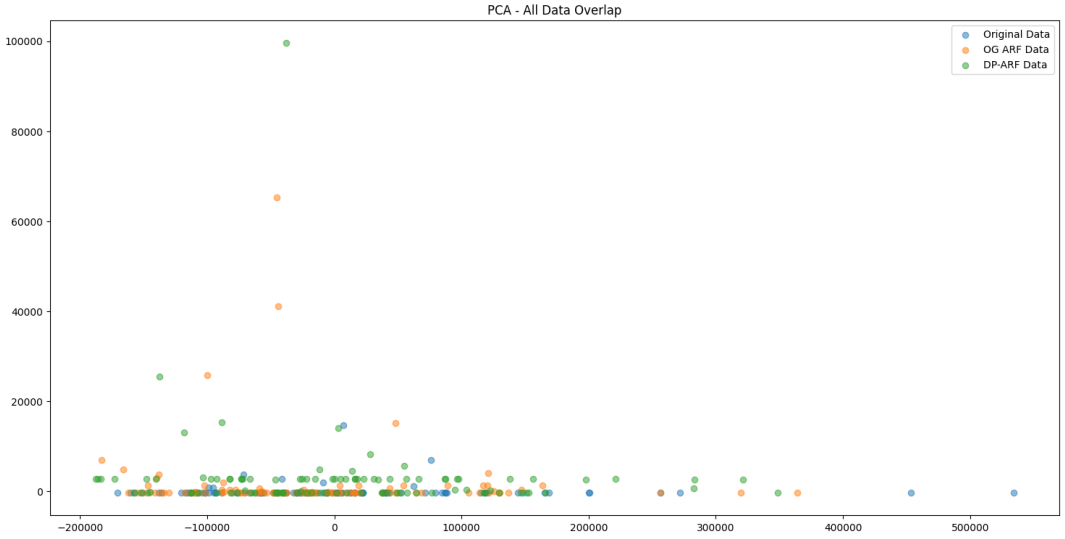

Figure 13: PCA analysis for Adult $\epsilon = 0.5$

Features	Original Data	OG ARF Data	DP ARF Data
Age	1.0000	1.0000	1.0000
Fnlwgt	-0.0766	-0.0748	-0.0573
Education Num	0.0365	0.0228	0.0435
Capital Gain	0.0776	0.0885	0.0853
Capital Loss	0.0577	0.0746	0.0598
Hours per Week	0.0687	0.0898	0.0723

Table 63: Correlation Matrices for Adult $\epsilon = 0.5$

Feature	OG ARF	DP ARF
Age	0.2970	0.5356
Fnlwgt	4836.6663	9087.6845
Education Num	0.3181	0.5221
Capital Gain	730.0260	2015.6014
Capital Loss	79.5174	170.4623
Hours per Week	1.5451	2.2142

Table 64: Wasserstein Distances for Adult $\epsilon = 0.5$

Feature	OG ARF	DP ARF
Workclass	3.2510	6493.7061
Education	15.9293	1736.4526
Marital Status	1.6512	1761.5229
Occupation	7.3814	1180.0485
Relationship	2.0247	1395.9324
Race	2.1653	3478.9120
Sex	0.3323	78.3628
Native Country	inf	4747.7375
Income	0.7360	465.4520

Table 65: Chi-Square Tests for Adult $\epsilon = 0.5$

Feature	OG ARF	DP ARF
Age	19.2967	20.1172
Fnlwgt	150145.8945	157823.5666
Education Num	3.6681	3.7928
Capital Gain	10679.0877	11814.0060
Capital Loss	555.6207	589.8089
Hours per Week	17.6909	18.8225

Table 66: RMSE for Adult $\epsilon = 0.5$

Feature	OG ARF	DP ARF
Age	15.4469	15.8346
Fnlwgt	113682.3628	117208.1162
Education Num	2.7879	2.9211
Capital Gain	2610.5686	3796.7102
Capital Loss	217.3906	309.4794
Hours per Week	13.0037	13.6893

Table 67: MAE for Adult $\epsilon = 0.5$

ARF	Value
OG	0.8364
DP	0.7312

Table 68: Classification Accuracy for Adult $\epsilon = 0.5$

ARF	Value
OG	239.2654
DP	268.4633

Table 69: Reconstruction Attack Error for Adult $\epsilon = 0.5$

For the adult dataset the balanced trade-off was found at epsilon 0.5 and hence its displayed above and the other results for epsilon 0.1 and 1 if necessary are displayed in the the Appendix section A.

The density plots shown in Figure 11 and Figure 12 show that it follows the original data distribution very closely for most of the features on the Adult dataset except age and capital loss though its not that deviated from the original. A lower epsilon would provide a higher level of privacy but starts diverging more from the original distribution.

Principal Component Analysis (PCA) performed on the datasets shown in Figure 13 at an epsilon of 0.5 expresses a good preservation of relationships between the variables.

Quantitative measures, such as Wasserstein distance, chi-square tests, and RMSE, show increasing errors and differences with increasing privacy. Also the model's performance, measured in terms of classification accuracy, was best for an epsilon of 0.5 in terms of difference from the OG arf shown in Table 68 thus confirming that this value offers a practically good trade off between privacy and utility.

Reconstruction attack error was good for all the values in the dp arf but the best one having a good trade off with utility was at 0.5 Table 69.

Overall, the DP-ARF was successful in generating realistic dataset from all the datasets (Insurance, Healthcare, Adult) all while preserving data utility and offering strong differential privacy guarantees.

6 Legal, Social, Ethical and Professional Issues

This project has adhered to the research and practices that match the ethical, legal, and professional standards set out by the British Computer Society (BCS) and The Institution of Engineering and Technology (IET). Public well being was at the forefront in this project as its main aim is to protect people's privacy through the use of differential privacy techniques. The methods outlined in the project support the rights and confidentiality of people represented within our datasets like the Adult dataset [18].

I have made a point of emphasizing the integrity regarding the possibilities and constraints that my algorithms can have reflecting upon my high standards for both honesty and responsibility.

This project fully supports these codes outlined above while dealing with technological innovation and ethical accountability. I meticulously reviewed each dataset to assure conformity with legal and ethical norms, as well as ensuring that my work was based on the principles of data protection and intellectual property. The project further mitigated risks of data security by using synthetic datasets in practice and encouraging trust in AI technologies. This thesis is not just a requirement for academic fulfillment but also speaks towards good use of technology in sensitive environments. With such work, I strived to set an example of how high-class data analysis should be balanced with strong ethical practices and data privacy to promote technology responsibly.

7 Conclusion

This research primarily looked into questions related to the integration of differential privacy into adversarial random forests in order to achieve data generation that is privatized. The research was centered on understanding the trade-offs of privacy against data utility, the challenges of operationalizing differential privacy in complex machine learning models.

The results demonstrated in section 5 show that DP can be embedded into ARF and will appropriately generate privacy protected synthetic data that strike a balance between privacy and utility. The DP-ARF model includes noise in the process of data generation to ensure privacy. However, this is sometimes at the expense of the statistical properties of the data. Comparative analyses with the OG ARF model showed that, while DP-ARF maintained similar trends in the distributions of generated data, privacy was significantly increased which made a slight degradation in utility performance metrics acceptable as outlined by the principles of differential privacy.

Concretely, this work contributes to the field of differential privacy and generative modeling by successfully implementing the differential privacy in the adversarial random forest framework [12]. This research has shed light on complexities and challenges involved in the integration of Differential Privacy and advanced machine learning models and has highlighted the necessary trade-offs in all these cases. The work is worth the effort since it extends the current understanding of privacy preserving synthetic data generation and constitutes the basis for future research in this area of study.

7.1 Future Work

Management of Privacy Budget

Construction of methods to track and optimize privacy budget consumption in future works can lead to sustainable and controlled usage of privacy resources. This would also lead to protection against the adaptive query attacks which is no explored in this work because the framework required to implement the adaptive queries pose significant complexity and manual intervention. Since, through the adaptive query attack, an attacker can exploit loopholes that occur during the query production process, thereby breaching the privacy of the user. Therefore, developing the DP-ARF model so as to protect it from such an attack would make it more robust and more applicable for real world scenario.

Automation of Privacy Parameters

The above privacy budget management can be further streamlined by automating the selection and tuning of privacy parameters. Thus, One way to make the model efficient and easy to use is to develop algorithms through which privacy budgets can be set in a more dynamic way according to data sensitivity and the context of their use.

Extension to Other Machine Learning Models

The extension of differential privacy techniques to other tree based machine learning models, apart from random forests, such as mentioned in section 3 can expand the applicability area of the privacy preserving data generation methods.

7.2 Lessons Learned

- **Importance of Prioritizing the Best Approach:**
  - One key lesson learned was the importance of prioritizing the best approach early on. Given the multitude of potential methods, it is essential to identify and focus on the most promising one to maximize efficiency and effectiveness.
- **Finding Alternatives Under Constraints:**
  - When implementation fails due to decisions taken during previous design stages, alternative approaches must be investigated quickly, this becomes very important when there are time constraints. The design of a project should be kept in mind from the beginning to strike a balance between troubleshooting to resolve current problems and moving forward to seek new techniques.
- **Project Structure:**
  - Having a good project structure and established goals from the start makes tracking progress simpler and prevents those useless changes. Engagement with supervisor and their guidance can further refine your approach.

Overall, The project outlines a successful integration of differential privacy into the ARF. We have also seen that many challenges still remain, particularly in balancing privacy and utility. However, the research and methodologies in this paper provide a solid grounding for future advancements. Continued advancements in this field will further enhance the capabilities and applications of privacy preserving generative modeling techniques.

References

[1] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2022.

[2] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.

[3] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, 10 2001.

[4] J. Friedman, "Greedy function approximation: A gradient boosting machine," *The Annals of Statistics*, vol. 29, 11 2000.

[5] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, ACM, Aug. 2016.

[6] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[7] P. Pathak, "Generative AI: A Technical Deep Dive into Security and Privacy Concerns — scribbledata.io." `https://www.scribbledata.io/blog/generative-ai-a-technical-deep-dive-into-security-and-privacy-concerns/`.

[8] L. Grinsztajn, E. Oyallon, and G. Varoquaux, "Why do tree-based models still outperform deep learning on tabular data?," 2022.

[9] V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci, "Deep neural networks and tabular data: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, p. 1–21, 2024.

[10] R. Nock and M. Guillame-Bert, "Generative trees: Adversarial and copycat," 2022.

[11] R. Nock and M. Guillame-Bert, "Generative forests," 2023.

[12] D. S. Watson, K. Blesch, J. Kapar, and M. N. Wright, "Adversarial random forests for density estimation and generative modeling," 2023.

[13] C. Dwork, "Differential privacy," in *Automata, Languages and Programming* (M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, eds.), (Berlin, Heidelberg), pp. 1–12, Springer Berlin Heidelberg, 2006.

[14] L. Xie, K. Lin, S. Wang, F. Wang, and J. Zhou, "Differentially private generative adversarial network," 2018.

[15] S. Maddock, G. Cormode, T. Wang, C. Maple, and S. Jha, "Federated boosted decision trees with differential privacy," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, ACM, Nov. 2022.

[16] K. Blesch and M. N. Wright, "arfpy: A python package for density estimation and generative modeling with adversarial random forests," 2023.

[17] A. Datta, "US Health Insurance Dataset," April 2020. Version 1. [Dataset; Kaggle]. Retrieved July 2024, from `https://www.kaggle.com/datasets/teertha/ushealthinsurancedataset/data?select=insurance.csv` The original source of this dataset is the 3rd edition of "Machine Learning with R", by Brett Lanz.

[18] B. Becker and R. Kohavi, "Adult." UCI Machine Learning Repository, 1996. DOI: https://doi.org/10.24432/C5XW20.

[19] P. Patil, "Healthcare dataset," 2024. Version 2. [Dataset; Kaggle]. Retrieved July 2024, from `https://www.kaggle.com/datasets/prasad22/healthcare-dataset`.

A Appendix

Appendix A: Source Code

Appendix A.1: DP-ARF Implementation Code

```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble._forest import _generate_unsampled_indices
import scipy
from arfpy import utils

class arf:
    """Implements Adversarial Random Forests (ARF) in python
    Usage:
    1. fit ARF model with arf()
    2. estimate density with arf.forde()
    3. generate data with arf.forge().

    :param x: Input data.
    :type x: pandas.Dataframe
    :param num_trees:  Number of trees to grow in each forest, defaults to 30
    :type num_trees: int, optional
    :param delta: Tolerance parameter. Algorithm converges when OOB accuracy is < 0.5 + `delta`, defaults to 0
    :type delta: float, optional
    :param max_iters: Maximum iterations for the adversarial loop, defaults to 10
    :type max_iters: int, optional
    :param early_stop: Terminate loop if performance fails to improve from one round to the next?, defaults to True
    :type early_stop: bool, optional
    :param verbose: Print discriminator accuracy after each round?, defaults to True
    :type verbose: bool, optional
    :param min_node_size: minimum number of samples in terminal node, defaults to 5
    :type min_node_size: int
    """

    def __init__(self, x, num_trees=30, delta=0, max_iters=10, early_stop=True, verbose=True, min_node_size=5, epsilon=1.0, **kwargs):
        # Assertions to ensure correct input types and values
        assert isinstance(x, pd.core.frame.DataFrame), f"expected pandas DataFrame as input, got:{type(x)}"
        assert len(set(list(x))) == x.shape[1], f"every column must have a unique column name"
        assert max_iters >= 0, f"negative number of iterations is not allowed: parameter max_iters must be >= 0"
        assert min_node_size > 0, f"minimum number of samples in terminal nodes (parameter min_node_size) must be greater than zero"
```

```
37        assert num_trees > 0, f"number of trees in the random forest (
    parameter num_trees) must be greater than zero"
38        assert 0 <= delta <= 0.5, f"parameter delta must be in range 0 <=
     delta <= 0.5"
39
40        # Initialize values
41        x_real = x.copy()
42        self.p = x_real.shape[1]
43        self.orig_colnames = list(x_real)
44        self.num_trees = num_trees
45
46        self.epsilon = epsilon  # Privacy budget added for dp_arf by
    Harsh Merchant
47
48        # Find object columns and convert to category
49        self.object_cols = x_real.dtypes == "object"
50        for col in list(x_real):
51            if self.object_cols[col]:
52                x_real[col] = x_real[col].astype('category')
53
54        # Find factor columns
55        self.factor_cols = x_real.dtypes == "category"
56
57        # Save factor levels
58        self.levels = {}
59        for col in list(x_real):
60            if self.factor_cols[col]:
61                self.levels[col] = x_real[col].cat.categories
62
63        # Recode factors to integers
64        for col in list(x_real):
65            if self.factor_cols[col]:
66                x_real[col] = x_real[col].cat.codes
67
68        # If no synthetic data provided, sample from marginals
69        x_synth = x_real.apply(lambda x: x.sample(frac=1).values)
70
71        # Merge real and synthetic data
72        x = pd.concat([x_real, x_synth])
73        y = np.concatenate([np.zeros(x_real.shape[0]), np.ones(x_real.
    shape[0])])
74        # real observations = 0, synthetic observations = 1
75
76        # Pass on x_real
77        self.x_real = x_real
78
79        # Fit initial RF model
80        clf_0 = RandomForestClassifier(oob_score=True, n_estimators=self.
    num_trees, min_samples_leaf=min_node_size, **kwargs)
81        clf_0.fit(x, y)
82
83        iters = 0
84
```

```
85          acc_0 = clf_0.oob_score_  # is accuracy directly
86          acc = [acc_0]
87
88          if verbose:
89              print(f'Initial accuracy is {acc_0}')
90
91          if acc_0 > 0.5 + delta and iters < max_iters:
92              converged = False
93              while not converged:  # Start adversarial loop
94                  # Get nodeIDs
95                  nodeIDs = clf_0.apply(self.x_real)  # dimension [
    terminalnode, tree]
96
97                  # Add observation ID to x_real
98                  x_real_obs = x_real.copy()
99                  x_real_obs['obs'] = range(0, x_real.shape[0])
100
101                 # Add observation ID to nodeIDs
102                 nodeIDs_pd = pd.DataFrame(nodeIDs)
103                 tmp = nodeIDs_pd.copy()
104                 tmp['obs'] = range(0, x_real.shape[0])
105                 tmp = tmp.melt(id_vars=['obs'], value_name="leaf",
    var_name="tree")
106
107                 # Match real data to trees and leafs (node id for tree)
108                 x_real_obs = pd.merge(left=x_real_obs, right=tmp, on=['
    obs'], sort=False)
109                 x_real_obs.drop('obs', axis=1, inplace=True)
110
111                 # Sample leafs
112                 tmp.drop("obs", axis=1, inplace=True)
113                 tmp = tmp.sample(x_real.shape[0], axis=0, replace=True)
114                 tmp = pd.Series(tmp.value_counts(sort=False), name='cnt')
    .reset_index()
115                 draw_from = pd.merge(left=tmp, right=x_real_obs, on=['
    tree', 'leaf'], sort=False)
116
117                 # Sample synthetic data from leaf
118                 grpd = draw_from.groupby(['tree', 'leaf'])
119                 x_synth = [grpd.get_group(ind).apply(lambda x: x.sample(n
    =grpd.get_group(ind)['cnt'].iloc[0], replace=True).values) for ind in
    grpd.indices]
120                 x_synth = pd.concat(x_synth).drop(['cnt', 'tree', 'leaf'
    ], axis=1)
121
122                 # Delete unnecessary objects
123                 del(nodeIDs, nodeIDs_pd, tmp, x_real_obs, draw_from)
124
125                 # Merge real and synthetic data
126                 x = pd.concat([x_real, x_synth])
127                 y = np.concatenate([np.zeros(x_real.shape[0]), np.ones(
    x_real.shape[0])])
128
```

```
                # Discriminator
                clf_1 = RandomForestClassifier(oob_score=True,
    n_estimators=self.num_trees, min_samples_leaf=min_node_size, **kwargs)
                clf_1.fit(x, y)

                # Update iters and check for convergence
                acc_1 = clf_1.oob_score_

                acc.append(acc_1)

                iters += 1
                plateau = True if early_stop and acc[iters] > acc[iters -
     1] else False
                if verbose:
                    print(f"Iteration number {iters} reached accuracy of
    {acc_1}.")
                if acc_1 <= 0.5 + delta or iters >= max_iters or plateau:
                    converged = True
                else:
                    clf_0 = clf_1
        self.clf = clf_0
        self.acc = acc

        # Pruning
        pred = self.clf.apply(self.x_real)
        for tree_num in range(0, self.num_trees):
            tree = self.clf.estimators_[tree_num]
            left = tree.tree_.children_left
            right = tree.tree_.children_right
            leaves = np.where(left < 0)[0]

            # Get leaves that are too small
            unique, counts = np.unique(pred[:, tree_num], return_counts=
    True)
            to_prune = unique[counts < min_node_size]

            # Also add leaves with 0 obs.
            to_prune = np.concatenate([to_prune, np.setdiff1d(leaves,
    unique)])

            while len(to_prune) > 0:
                for tp in to_prune:
                    # Find parent
                    parent = np.where(left == tp)[0]
                    if len(parent) > 0:
                        # Left child
                        left[parent] = right[parent]
                    else:
                        # Right child
                        parent = np.where(right == tp)[0]
                        right[parent] = left[parent]
                # Prune again if child was pruned
                to_prune = np.where(np.in1d(left, to_prune))[0]
```

```
177
178     def forde(self, dist="truncnorm", oob=False, alpha=0):
179         """This part is for density estimation (FORDE)
180
181         :param dist: Distribution to use for density estimation of
    continuous features. Distributions implemented so far: "truncnorm",
    defaults to "truncnorm"
182         :type dist: str, optional
183         :param oob: Only use out-of-bag samples for parameter estimation?
     If `True`, `x` must be the same dataset used to train `arf`, defaults
     to False
184         :type oob: bool, optional
185         :param alpha: Optional pseudocount for Laplace smoothing of
    categorical features. This avoids zero-mass points when test data fall
     outside the support of training data. Effectively parametrizes a flat
     Dirichlet prior on multinomial likelihoods, defaults to 0
186         :type alpha: float, optional
187         :return: Return parameters for the estimated density.
188         :rtype: dict
189         """
190
191         self.dist = dist
192         self.oob = oob
193         self.alpha = alpha
194
195         # Get terminal nodes for all observations
196         pred = self.clf.apply(self.x_real)
197
198         # If OOB, use only OOB trees
199         if self.oob:
200             for tree in range(self.num_trees):
201                 idx_oob = np.isin(range(self.x_real.shape[0]),
    _generate_unsampled_indices(self.clf.estimators_[tree].random_state,
    self.x_real.shape[0]))
202                 pred[np.invert(idx_oob), tree] = -1
203
204         # Compute leaf bounds and coverage
205         bnds = pd.concat([utils.bnd_fun(tree=j, p=self.p, forest=self.clf
    , feature_names=self.orig_colnames) for j in range(self.num_trees)])
206         bnds['f_idx'] = bnds.groupby(['tree', 'leaf']).ngroup()
207
208         bnds_2 = pd.DataFrame()
209         for t in range(self.num_trees):
210             unique, freq = np.unique(pred[:, t], return_counts=True)
211             vv = pd.concat([pd.Series(unique, name='leaf'), pd.Series(
    freq/pred.shape[0], name='cvg')], axis=1)
212             zz = bnds[bnds['tree'] == t]
213             bnds_2 = pd.concat([bnds_2, pd.merge(left=vv, right=zz, on=['
    leaf'])])
214         bnds = bnds_2
215         del(bnds_2)
216
217         # Set coverage for nodes with single observations to zero
```

```
218         if np.invert(self.factor_cols).any():
219             bnds.loc[bnds['cvg'] == 1 / pred.shape[0], 'cvg'] = 0
220 
221         # No parameters to learn for zero coverage leaves - drop zero
    coverage nodes
222         bnds = bnds[bnds['cvg'] > 0]
223 
224         # Rename leafs to nodeids
225         bnds.rename(columns={'leaf': 'nodeid'}, inplace=True)
226 
227         # Save bounds to later use coverage for drawing new samples
228         self.bnds = bnds
229         # Fit continuous distribution in all terminal nodes
230         self.params = pd.DataFrame()
231         if np.invert(self.factor_cols).any():
232             for tree in range(self.num_trees):
233                 dt = self.x_real.loc[:, np.invert(self.factor_cols)].copy
    ()
234                 dt["tree"] = tree
235                 dt["nodeid"] = pred[:, tree]
236                 # Merge bounds and make it long format
237                 long = pd.merge(right=bnds[['tree', 'nodeid', 'variable',
     'min', 'max', 'f_idx']], left=pd.melt(dt[dt["nodeid"] >= 0], id_vars
    =["tree", "nodeid"]), on=['tree', 'nodeid', 'variable'], how='left')
238                 # Get distribution parameters
239                 if self.dist == "truncnorm":
240                     # Calculate sensitivity and add noise to continous
    variables for dp_arf by Harsh Merchant
241                     observed_min = long.groupby(['tree', 'nodeid', '
    variable'])['value'].min().reset_index()
242                     observed_max = long.groupby(['tree', 'nodeid', '
    variable'])['value'].max().reset_index()
243                     sensitivity = (observed_max['value'] - observed_min['
    value']).abs() / long.groupby(['tree', 'nodeid', 'variable'])['value'
    ].size().reset_index(drop=True)
244                     sensitivity.replace(0, 1e-6, inplace=True)  # Avoid
    zero sensitivity
245 
246                     res = long.groupby(['tree', "nodeid", "variable"],
    as_index=False).agg(mean=("value", "mean"), sd=("value", "std"), min=(
    "min", "min"), max=("max", "max"))
247 
248                     # Add Laplace noise for differential privacy
249                     scale = sensitivity / self.epsilon  # Scale of
    Laplace noise
250                     mean_noise = np.random.laplace(0, scale, size=res["
    mean"].shape)
251 
252                     res["mean"] += mean_noise  # Adding noise to mean
253 
254                     # Recalculate standard deviation based on new noisy
    mean
255                     for idx, row in res.iterrows():
```

```
256                         variable = row["variable"]
257                         tree = row["tree"]
258                         nodeid = row["nodeid"]
259                         group = long[(long["variable"] == variable) & (
    long["tree"] == tree) & (long["nodeid"] == nodeid)]
260                         noisy_mean = row["mean"]
261                         recalculated_sd = np.sqrt(np.mean((group["value"]
     - noisy_mean)**2))
262                         res.at[idx, "sd"] = recalculated_sd
263                     # Calculate sensitivity for dp_arf by Harsh Merchant
264 
265                 else:
266                     raise ValueError('unknown distribution, make sure to
    enter a valid value for dist')
267                 self.params = pd.concat([self.params, res])
268 
269         # Get class probabilities in all terminal nodes
270         self.class_probs = pd.DataFrame()
271         if self.factor_cols.any():
272             for tree in range(self.num_trees):
273                 dt = self.x_real.loc[:, self.factor_cols].copy()
274                 dt["tree"] = tree
275                 dt["nodeid"] = pred[:, tree]
276                 dt = pd.melt(dt[dt["nodeid"] >= 0], id_vars=["tree", "
    nodeid"])
277                 long = pd.merge(left=dt, right=bnds, on=['tree', 'nodeid'
    , 'variable'])
278                 long['count_var'] = long.groupby(['tree', 'nodeid', '
    variable'])['variable'].transform('count')
279                 long['count_var_val'] = long.groupby(['tree', 'nodeid', '
    variable', 'value'])['variable'].transform('count')
280                 long.drop_duplicates(inplace=True)
281                 if self.alpha == 0:
282 
283                     # Calculate initial probabilities
284                     long['prob'] = long['count_var_val'] / long['
    count_var']
285 
286                     # Changes for dp_arf by Harsh Merchant
287 
288                     # Use these initial probabilities as the utility
    scores
289                     scores = long['prob']
290 
291                     scale = 1 / self.epsilon  # Sensitivity is 1 for
    counting queries
292 
293                     # Apply the exponential mechanism
294                     exp_noise = np.exp(self.epsilon * scores / 2)
295 
296                     long['prob'] = exp_noise  # Properly scaled
297 
298                     # Normalize within groups to form a valid probability
```

```
 distribution
                long['prob'] /= long.groupby(['tree', 'nodeid', '
variable'])['prob'].transform('sum')

                # Changes for dp_arf by Harsh Merchant

            else:
                long['k'] = long.groupby(['variable'])['value'].
transform('nunique')
                long.loc[long['min'] == float('-inf'), 'min'] = 0.5 -
 1
                long.loc[long['max'] == float('inf'), 'max'] = long['
k'] + 0.5 - 1
                long.loc[round(long['min'] % 1, 2) != 0.5, 'min'] =
long['min'] - 0.5
                long.loc[round(long['max'] % 1, 2) != 0.5, 'max'] =
long['max'] + 0.5   # Added Code{correction min to max} for same array
 length error - Harsh Merchant
                long['k'] = long['max'] - long['min']
                tmp = long[['f_idx', 'tree', "nodeid", 'variable', '
min', 'max']].copy()
                tmp['rep_min'] = tmp['min'] + 0.5
                tmp['rep_max'] = tmp['max'] - 0.5
                tmp['levels'] = tmp.apply(lambda row: list(range(int(
row['rep_min']), int(row['rep_max'] + 1))), axis=1)
                tmp = tmp.explode('levels')
                # Added Code for same array length error - Harsh
Merchant
                original_levels = {key: values.copy() for key, values
 in self.levels.items()}
                # Step 1: Find the maximum length of the lists
                max_length = max(len(values) for values in self.
levels.values())

                # Step 2: Fill shorter lists with None to match the
maximum length
                for key, values in self.levels.items():
                    if len(values) < max_length:
                        self.levels[key] = values.tolist() + [None] *
 (max_length - len(values))
                # Added Code for same array length error - Harsh
Merchant

                cat_val = pd.DataFrame(self.levels).melt().dropna() #
 Added Code{.dropna()} for same array length error - Harsh Merchant
                self.levels = original_levels # Added Code for same
array length error - Harsh Merchant
                cat_val['value'] = cat_val.groupby('variable').
cumcount() # Added Code for same array length error - Harsh Merchant
                cat_val['levels'] = cat_val['value']

                tmp = pd.merge(left=tmp, right=cat_val, on=['variable
', 'levels'])[['variable', 'f_idx', 'tree', "nodeid", 'value']]
```

```
332
333                     tmp = pd.merge(left=tmp, right=long[['f_idx', '
    variable', 'tree', "nodeid", 'count_var', 'k']], on=['f_idx', "nodeid"
    , 'variable', 'tree'])
334                     long = pd.merge(left=tmp, right=long, on=['f_idx', '
    tree', "nodeid", 'variable', 'value', 'count_var', 'k'], how='left')
335                     long.loc[long['count_var_val'].isna(), 'count_var_val
    '] = 0
336                     long = long[['f_idx', 'tree', "nodeid", 'variable', '
    value', 'count_var_val', 'count_var', 'k']].drop_duplicates()
337                     long['prob'] = (long['count_var_val'] + self.alpha) /
     (long['count_var'] + self.alpha*long['k'])
338
339                     # Changes for dp_arf by Harsh Merchant
340                     scores = long['prob']
341                     scale = 1 / self.epsilon  # Sensitivity is 1 for
    counting queries
342                     exp_noise = np.exp(self.epsilon * scores / 2)
343
344                     long['prob'] = exp_noise  # Properly scaled
345
346                     long['prob'] /= long.groupby(['tree', 'nodeid', '
    variable'])['prob'].transform('sum')
347                     # Changes for dp_arf by Harsh Merchant
348
349                     # long['value'] = long['value'].astype('int8') # Code
     Commented for removing negative values because conversion > 128 for
    int8  - Harsh Merchant
350
351                 long = long[['f_idx', 'tree', "nodeid", 'variable', '
    value', 'prob']]
352                 self.class_probs = pd.concat([self.class_probs, long])
353             return {"cnt": self.params, "cat": self.class_probs, "forest"
    : self.clf, "meta": pd.DataFrame(data={"variable": self.orig_colnames,
     "family": self.dist})}
354
355     def forge(self, n):
356         """This part is for data generation (FORGE)
357
358         :param n: Number of synthetic samples to generate.
359         :type n: int
360         :return: Returns generated data.
361         :rtype: pandas.DataFrame
362         """
363
364         try:
365             getattr(self, 'bnds')
366         except AttributeError:
367             raise AttributeError('need density estimates to generate data
     -- run .forde() first!')
368
369         # Sample new observations and get their terminal nodes
370         # Draw random leaves with probability proportional to coverage
```

```
371         unique_bnds = self.bnds[['tree', 'nodeid', 'cvg']].
    drop_duplicates()
372         draws = np.random.choice(a=range(unique_bnds.shape[0]), p=
    unique_bnds['cvg'] / self.num_trees, size=n)
373         sampled_trees_nodes = unique_bnds[['tree', 'nodeid']].iloc[draws
    ,].reset_index(drop=True).reset_index().rename(columns={'index': 'obs'
    })
374 
375         # Get distributions parameters for each new obs.
376         if np.invert(self.factor_cols).any():
377             obs_params = pd.merge(sampled_trees_nodes, self.params, on=["
    tree", "nodeid"]).sort_values(by=['obs'], ignore_index=True)
378 
379         # Get probabilities for each new obs.
380         if self.factor_cols.any():
381             obs_probs = pd.merge(sampled_trees_nodes, self.class_probs,
    on=["tree", "nodeid"]).sort_values(by=['obs'], ignore_index=True)
382 
383         # Sample new data from mixture distribution over trees
384         data_new = pd.DataFrame(index=range(n), columns=range(self.p))
385         for j in range(self.p):
386             colname = self.orig_colnames[j]
387 
388             if self.factor_cols[j]:
389                 # Factor columns: Multinomial distribution
390                 data_new.iloc[:, j] = obs_probs[obs_probs["variable"] ==
    colname].groupby("obs").sample(weights="prob")["value"].reset_index(
    drop=True)
391 
392             else:
393                 # Continuous columns: Match estimated distribution
    parameters with r...() function
394                 if self.dist == "truncnorm":
395                     # sample from normal distribution, only here for
    debugging
396                     # data_new.loc[:, j] = np.random.normal(obs_params.
    loc[obs_params["variable"] == colname, "mean"], obs_params.loc[
    obs_params["variable"] == colname, "sd"], size = n)
397 
398                     # sample from truncated normal distribution
399                     # note: if sd == 0, truncnorm will return location
    parameter -> this is desired; if we have
400                     # all obs. in that leave having the same value, we
    sample a new obs. with exactly that value as well
401                     myclip_a = obs_params.loc[obs_params["variable"] ==
    colname, "min"]
402                     myclip_b = obs_params.loc[obs_params["variable"] ==
    colname, "max"]
403                     myloc = obs_params.loc[obs_params["variable"] ==
    colname, "mean"]
404                     myscale = obs_params.loc[obs_params["variable"] ==
    colname, "sd"]
405                     data_new.isetitem(j, scipy.stats.truncnorm(a =(
```

```
    myclip_a - myloc) / myscale,b = (myclip_b - myloc) / myscale, loc =
    myloc , scale = myscale ).rvs(size = n))
406                 del(myclip_a,myclip_b,myloc,myscale)
407             else:
408                 raise ValueError('Other distributions not yet
    implemented')
409
410         # Use original column names
411         data_new = data_new.set_axis(self.orig_colnames, axis=1, copy=
    False)
412
413         # Convert categories back to category
414         for col in self.orig_colnames:
415             if self.factor_cols[col]:
416                 data_new[col] = pd.Categorical.from_codes(data_new[col].
    astype(int), categories=self.levels[col])
417
418         # Convert object columns back to object
419         for col in self.orig_colnames:
420             if self.object_cols[col]:
421                 data_new[col] = data_new[col].astype("object")
422
423         # Return newly sampled data
424         return data_new
```

Appendix A.2: Modifications for alpha parameter in ARF Code

The changes below were made to the original arfpy code because passing the alpha parameter in the forde function of the original arfpy resulted in a ValueError. This error occurred because the arrays in the self.levels attribute were not all the same length.

These changes work on datasets with small to medium-sized distinct categorical feature values. For datasets with a large number of distinct categorical feature values, a High-Performance Computing Cluster would be required. Otherwise, it would throw a MemoryError due to the poor handling of categorical feature value encoding. Thus, this is a temporary solution and it requires careful calibration and ample time to tweak to make it fully functional as originally intended.

```
if self.alpha == 0:
          long['prob'] = long['count_var_val'] / long['count_var']
        else:
          # Define the range of each variable in each leaf
          long['k'] = long.groupby(['variable'])['value'].transform('nunique')
          long.loc[long['min'] == float('-inf') , 'min'] = 0.5 - 1
          long.loc[long['max'] == float('inf') , 'max'] = long['k'] + 0.5 - 1
          long.loc[round(long['min'] % 1,2) != 0.5 , 'min'] = long['min'] - 0.5
          long.loc[round(long['max'] % 1,2) != 0.5 , 'max'] = long['max'] + 0.5   # Added Code{correction min to max} for same array length error - Harsh Merchant
          long['k'] = long['max'] - long['min']
          # Enumerate each possible leaf-variable-value combo
          tmp = long[['f_idx','tree', "nodeid", 'variable', 'min','max']].copy()
          tmp['rep_min'] = tmp['min'] + 0.5
          tmp['rep_max'] = tmp['max'] - 0.5
          tmp['levels'] = tmp.apply(lambda row:  list(range(int(row['rep_min']), int(row['rep_max'] + 1))), axis=1)
          tmp = tmp.explode('levels')

          # Added Code for same array length error - Harsh Merchant
          original_levels = {key: values.copy() for key, values in self.levels.items()}
          # Step 1: Find the maximum length of the lists
          max_length = max(len(values) for values in self.levels.values())

          # Step 2: Fill shorter lists with None to match the maximum length
          for key, values in self.levels.items():
              if len(values) < max_length:
                  self.levels[key] = values.tolist() + [None] * (max_length - len(values))
          # Added Code for same array length error - Harsh Merchant

          cat_val = pd.DataFrame(self.levels).melt().dropna() # Added
```

```
    Code{.dropna()} for same array length error - Harsh Merchant
30          self.levels = original_levels # Added Code for same array
    length error - Harsh Merchant
31          cat_val['value'] = cat_val.groupby('variable').cumcount() #
    Added Code for same array length error - Harsh Merchant
32
33          cat_val['levels'] = cat_val['value']
34          tmp =  pd.merge(left = tmp, right = cat_val, on = ['variable',
    'levels'])[['variable', 'f_idx','tree', "nodeid",'value']]
35
36          # populate count, k
37          tmp = pd.merge(left = tmp, right = long[['f_idx', 'variable', '
    tree', "nodeid",'count_var', 'k']], on = ['f_idx', "nodeid", 'variable
    ', 'tree'])
38
39          # Merge with long, set val_count = 0 for possible but
    unobserved levels
40          long = pd.merge(left = tmp, right = long, on = ['f_idx','tree',
    "nodeid",  'variable','value','count_var','k'], how = 'left')
41
42          long.loc[long['count_var_val'].isna(), 'count_var_val'] = 0
43
44          long = long[['f_idx','tree',"nodeid",  'variable', 'value', '
    count_var_val', 'count_var', 'k']].drop_duplicates()
45          # Compute posterior probabilities
46          long['prob'] = (long['count_var_val'] + self.alpha) / (long['
    count_var'] + self.alpha*long['k'])
47
48          # long['value'] = long['value'].astype('int8') # Code Commented
     for removing negative values because conversion > 128 for int8  -
    Harsh Merchant
49
50      long = long[['f_idx','tree', "nodeid", 'variable', 'value','prob'
    ]]
51
52      self.class_probs = pd.concat([self.class_probs, long])
53   return {"cnt": self.params, "cat": self.class_probs,
54           "forest": self.clf, "meta" : pd.DataFrame(data={"variable":
    self.orig_colnames, "family": self.dist})}
```

Appendix A.3: Evaluation Script for Insurance Dataset

```
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error
from sklearn.metrics import accuracy_score, roc_auc_score,
    precision_score, recall_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.neighbors import NearestNeighbors
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from scipy.stats import chisquare, wasserstein_distance
import seaborn as sns
import sys
import os

# Add the parent directory to the system path
sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath('dp_arf.
    py'))))
from dp_arf import arf as DP_ARF  # Import the DP_ARF class
from arfpy import arf as OG_ARF  # Import the OG ARF class without
    differential privacy

def load_new_dataset(file_path):
    return pd.read_csv(file_path)

def plot_density_distribution(data1, data2, data3, columns, sample_size
    =1000):
    # Sample the data if it exceeds the sample size
    if data1.shape[0] > sample_size:
        data1_sample = data1.sample(n=sample_size)
        data2_sample = data2.sample(n=sample_size)
        data3_sample = data3.sample(n=sample_size)
    else:
        data1_sample = data1
        data2_sample = data2
        data3_sample = data3

    fig, axs = plt.subplots(2, 2, figsize=(14, 10))
    axs = axs.flatten()

    plot_count = 0

    for column in columns:
        if plot_count >= 4:
            break
```

```
48
49          if pd.api.types.is_numeric_dtype(data1_sample[column]):
50              data1_sample[column].plot(kind='density', ax=axs[plot_count],
    label='Original Data', legend=True)
51              data2_sample[column].plot(kind='density', ax=axs[plot_count],
    label='OG ARF Data', legend=True)
52              data3_sample[column].plot(kind='density', ax=axs[plot_count],
    label='DP-ARF Data', legend=True)
53              axs[plot_count].set_title(f'Density Distribution for {column}
    ')
54              axs[plot_count].legend()
55              plot_count += 1
56          else:
57              print(f"Skipping column {column} as it is not numeric.")
58
59      plt.tight_layout()
60      plt.show()
61
62  def plot_pca(data1, data2, data3, sample_size=1000):
63      if data1.shape[0] > sample_size:
64          data1_sample = data1.sample(n=sample_size)
65          data2_sample = data2.sample(n=sample_size)
66          data3_sample = data3.sample(n=sample_size)
67      else:
68          data1_sample = data1
69          data2_sample = data2
70          data3_sample = data3
71
72      # Select only numeric columns for PCA
73      data1_numeric = data1_sample.select_dtypes(include=[np.number])
74      data2_numeric = data2_sample.select_dtypes(include=[np.number])
75      data3_numeric = data3_sample.select_dtypes(include=[np.number])
76
77      # Perform PCA
78      pca = PCA(n_components=2)
79      data1_pca = pca.fit_transform(data1_numeric)
80      data2_pca = pca.transform(data2_numeric)
81      data3_pca = pca.transform(data3_numeric)
82
83      fig, ax = plt.subplots(figsize=(10, 8))
84
85      ax.scatter(data1_pca[:, 0], data1_pca[:, 1], alpha=0.5, label='
    Original Data')
86      ax.scatter(data2_pca[:, 0], data2_pca[:, 1], alpha=0.5, label='OG ARF
     Data')
87      ax.scatter(data3_pca[:, 0], data3_pca[:, 1], alpha=0.5, label='DP-ARF
     Data')
88      ax.legend()
89      ax.set_title('PCA - All Data Overlap')
90
91      plt.tight_layout()
92      plt.show()
93
```

```
94 def calculate_additional_metrics(original_data, og_arf_data, dp_arf_data):
95     metrics = {}
96 
97     original_data_sampled = original_data.sample(n=og_arf_data.shape[0])
98 
99     # Calculate Wasserstein Distance for numeric columns
100    for col in original_data.select_dtypes(include=['int64', 'float64']).columns:
101        wasserstein_og = wasserstein_distance(original_data_sampled[col], og_arf_data[col])
102        wasserstein_dp = wasserstein_distance(original_data_sampled[col], dp_arf_data[col])
103        metrics[f'Wasserstein Distance {col}'] = {'OG ARF': wasserstein_og, 'DP ARF': wasserstein_dp}
104 
105    # Calculate Chi-Square Test for categorical columns
106    for col in original_data.select_dtypes(include=['category', 'object']).columns:
107        original_counts = original_data_sampled[col].value_counts().sort_index()
108        og_arf_counts = og_arf_data[col].value_counts().sort_index()
109        dp_arf_counts = dp_arf_data[col].value_counts().sort_index()
110 
111        categories = original_counts.index.union(og_arf_counts.index).union(dp_arf_counts.index)
112        original_counts = original_counts.reindex(categories, fill_value=0)
113        og_arf_counts = og_arf_counts.reindex(categories, fill_value=0)
114        dp_arf_counts = dp_arf_counts.reindex(categories, fill_value=0)
115 
116        chi2_stat_og, chi2_pvalue_og = chisquare(original_counts, og_arf_counts)
117        chi2_stat_dp, chi2_pvalue_dp = chisquare(original_counts, dp_arf_counts)
118        metrics[f'Chi-Square Test {col}'] = {'OG ARF': chi2_stat_og, 'DP ARF': chi2_stat_dp}
119 
120    # Calculate RMSE and MAE for numeric columns
121    for col in original_data.select_dtypes(include=['int64', 'float64']).columns:
122        rmse_og = np.sqrt(mean_squared_error(original_data_sampled[col], og_arf_data[col]))
123        rmse_dp = np.sqrt(mean_squared_error(original_data_sampled[col], dp_arf_data[col]))
124        mae_og = np.mean(np.abs(original_data_sampled[col] - og_arf_data[col]))
125        mae_dp = np.mean(np.abs(original_data_sampled[col] - dp_arf_data[col]))
126        metrics[f'RMSE {col}'] = {'OG ARF': rmse_og, 'DP ARF': rmse_dp}
127        metrics[f'MAE {col}'] = {'OG ARF': mae_og, 'DP ARF': mae_dp}
128 
129    return metrics
```

```
130
131 # Calculate correlation matrices for datasets
132 def get_correlation_matrices(data1, data2, data3):
133     corr_matrices = {
134         "Original Data": data1.corr(),
135         "OG ARF Data": data2.corr(),
136         "DP-ARF Data": data3.corr()
137     }
138     return corr_matrices
139
140 # Evaluate classification model on synthetic data fitted on original data
141 def model_evaluation_on_classification(original_data, synthetic_data,
    target_col='class'):
142     og = original_data.copy()
143     dp = synthetic_data.copy()
144     og[target_col] = (og['smoker'] == 'yes').astype(int)
145     dp[target_col] = (dp['smoker'] == 'yes').astype(int)
146
147     X_orig = og.drop(columns=[target_col])
148     y_orig = og[target_col]
149
150     X_synth = dp.drop(columns=[target_col])
151     y_synth = dp[target_col]
152
153     categorical_features = X_orig.select_dtypes(include=['object', '
    category']).columns
154     numeric_features = X_orig.select_dtypes(include=[np.number]).columns
155
156     preprocessor = ColumnTransformer(
157         transformers=[
158             ('num', 'passthrough', numeric_features),
159             ('cat', OneHotEncoder(handle_unknown='ignore'),
    categorical_features)
160         ])
161
162     X_encoded_orig = preprocessor.fit_transform(X_orig)
163     X_encoded_synth = preprocessor.transform(X_synth)
164
165     clf = LogisticRegression(max_iter=2000)
166     clf.fit(X_encoded_orig, y_orig)
167
168     return clf.score(X_encoded_synth, y_synth)
169
170
171 # Preprocessing function to encode categorical variables
172 def preprocess_data(df):
173     categorical_features = df.select_dtypes(include=['category']).columns
174     numeric_features = df.select_dtypes(include=[np.number]).columns
175
176     preprocessor = ColumnTransformer(
177         transformers=[
178             ('num', 'passthrough', numeric_features),
179             ('cat', OneHotEncoder(handle_unknown='ignore'),
```

```
    categorical_features)
        ])

    return preprocessor.fit_transform(df), preprocessor

# Reconstruction Attack
def reconstruction_attack(original_data, synthetic_data):
    X_orig, orig_preprocessor = preprocess_data(original_data.drop(columns=['charges']))
    X_synth, synth_preprocessor = preprocess_data(synthetic_data.drop(columns=['charges']))

    nn = NearestNeighbors(n_neighbors=3)
    nn.fit(X_synth)
    distances, indices = nn.kneighbors(X_orig)

    return distances.mean()

# Post process synthetic data to ensure valid values
def post_process_synthetic_data(data):
    data['age'] = data['age'].astype('int')
    data['children'] = data['children'].astype('int')
    data.loc[data['children'] < 0, 'children'] = 0
    mean_bmi = data['bmi'][data['bmi'] >= 0].mean()
    data.loc[data['bmi'] < 0, 'bmi'] = mean_bmi
    data['bmi'] = data['bmi'].round(2)
    mean_charges = data['charges'][data['charges'] >= 0].mean()
    data.loc[data['charges'] < 0, 'charges'] = mean_charges
    data['charges'] = data['charges'].round(2)
    return data


# Main Script
file_path = '..\\datasets\\insurance.csv'
data = load_new_dataset(file_path)

# Generate synthetic data using OG ARF model
arf_model = OG_ARF.arf(data)
arf_model.forde()
og_arf_data = arf_model.forge(data.shape[0])
og_arf_data = post_process_synthetic_data(og_arf_data)

# Generate synthetic data using DP-ARF model
dp_arf_model = DP_ARF(data, epsilon=0.5)
dp_arf_model.forde()
dp_arf_data = dp_arf_model.forge(data.shape[0])
dp_arf_data = post_process_synthetic_data(dp_arf_data)

# Print the first few rows of the datasets
print('original data: ', data.head())
print('og arf data: ', og_arf_data.head())
print('dp arf data: ', dp_arf_data.head())

```

```
# Identify numeric and categorical columns
numeric_columns = data.select_dtypes(include=[np.number]).columns.tolist()
categorical_columns = data.select_dtypes(include=['category', 'object']).columns.tolist()

print("Numeric columns:", numeric_columns)
print("Categorical columns:", categorical_columns)

# Plot density distributions
plot_density_distribution(data, og_arf_data, dp_arf_data, numeric_columns, sample_size=100)

# Plot PCA results
numeric_data = data.select_dtypes(include=[np.number])
numeric_og_arf_data = og_arf_data.select_dtypes(include=[np.number])
numeric_dp_arf_data = dp_arf_data.select_dtypes(include=[np.number])

plot_pca(numeric_data, numeric_og_arf_data, numeric_dp_arf_data, sample_size=100)

# Calculate and print correlation matrices
correlation_matrices = get_correlation_matrices(numeric_data, numeric_og_arf_data, numeric_dp_arf_data)
for key, value in correlation_matrices.items():
    print(f'{key} Correlation Matrix:')
    print(value)

# Calculate and print additional metrics
additional_metrics = calculate_additional_metrics(data, og_arf_data, dp_arf_data)
for metric, values in additional_metrics.items():
    print(f'{metric}:')
    for method, value in values.items():
        print(f'  {method}: {value}')

# Evaluate classification accuracy
accuracy_og = model_evaluation_on_classification(data, og_arf_data)
accuracy_dp = model_evaluation_on_classification(data, dp_arf_data)
print(f'Classification Accuracy (OG ARF): {accuracy_og}')
print(f'Classification Accuracy (DP ARF): {accuracy_dp}')

# Evaluate Attacks

# Evaluate reconstruction attack on og arf data
reconstruction_error = reconstruction_attack(data, og_arf_data)
print(f'Reconstruction Attack Error OG ARF (Mean Distance): {reconstruction_error}')

# Evaluate reconstruction attack on dp arf data
reconstruction_error = reconstruction_attack(data, dp_arf_data)
print(f'Reconstruction Attack Error DP ARF (Mean Distance): {reconstruction_error}')
```

Appendix A.4: Evaluation Script for Healthcare Dataset

```
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error, accuracy_score,
    roc_auc_score, precision_score, recall_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.neighbors import NearestNeighbors
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from scipy.stats import chisquare, wasserstein_distance
import seaborn as sns
import sys
import os

# Add the parent directory to the system path
sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath('dp_arf.
    py'))))
from dp_arf import arf as DP_ARF  # Import the DP_ARF class
from arfpy import arf as OG_ARF  # Import the OG ARF class without
    differential privacy

def load_new_dataset(file_path):
    return pd.read_csv(file_path)

def plot_density_distribution(data1, data2, data3, columns, sample_size
    =1000):
    # Sample the data if it exceeds the sample size
    if data1.shape[0] > sample_size:
        data1_sample = data1.sample(n=sample_size)
        data2_sample = data2.sample(n=sample_size)
        data3_sample = data3.sample(n=sample_size)
    else:
        data1_sample = data1
        data2_sample = data2
        data3_sample = data3

    # Initialize plot count
    plot_count = 0

    # Create subplots iteratively
    for i in range(0, len(columns), 4):
        num_cols = min(4, len(columns) - i)
        if num_cols == 1:
            fig, axs = plt.subplots(1, 1, figsize=(7, 5))
            axs = [axs]
```

```
48          elif num_cols == 2:
49              fig, axs = plt.subplots(1, 2, figsize=(14, 5))
50          elif num_cols == 3:
51              fig, axs = plt.subplots(2, 2, figsize=(14, 10))
52              axs[1, 1].remove()  # Remove the unused subplot
53              axs = [axs[0, 0], axs[0, 1], axs[1, 0]]
54          else:  # num_cols == 4
55              fig, axs = plt.subplots(2, 2, figsize=(14, 10))
56              axs = axs.flatten()  # Flatten the 2D array of axes to easily
     index them
57
58          for j, column in enumerate(columns[i:i+num_cols]):
59              if pd.api.types.is_numeric_dtype(data1_sample[column]):
60                  # Plot the density distributions on the corresponding
    subplot
61                  data1_sample[column].plot(kind='density', ax=axs[j],
    label='Original Data', legend=True)
62                  data2_sample[column].plot(kind='density', ax=axs[j],
    label='OG ARF Data', legend=True)
63                  data3_sample[column].plot(kind='density', ax=axs[j],
    label='DP-ARF Data', legend=True)
64                  axs[j].set_title(f'Density Distribution for {column}')
65                  axs[j].legend()
66              else:
67                  axs[j].set_title(f'Skipping column {column} as it is not
    numeric.')
68                  axs[j].axis('off')
69
70          # Adjust layout
71          plt.tight_layout()
72          plt.show()
73
74  def plot_pca(data1, data2, data3, sample_size=1000):
75      if data1.shape[0] > sample_size:
76          data1_sample = data1.sample(n=sample_size)
77          data2_sample = data2.sample(n=sample_size)
78          data3_sample = data3.sample(n=sample_size)
79      else:
80          data1_sample = data1
81          data2_sample = data2
82          data3_sample = data3
83
84      # Select only numeric columns for PCA
85      data1_numeric = data1_sample.select_dtypes(include=[np.number])
86      data2_numeric = data2_sample.select_dtypes(include=[np.number])
87      data3_numeric = data3_sample.select_dtypes(include=[np.number])
88
89      # Perform PCA
90      pca = PCA(n_components=2)
91      data1_pca = pca.fit_transform(data1_numeric)
92      data2_pca = pca.transform(data2_numeric)
93      data3_pca = pca.transform(data3_numeric)
94
```

```
    fig, ax = plt.subplots(figsize=(10, 8))  # Adjusted grid size

    ax.scatter(data1_pca[:, 0], data1_pca[:, 1], alpha=0.5, label='Original Data')
    ax.scatter(data2_pca[:, 0], data2_pca[:, 1], alpha=0.5, label='OG ARF Data')
    ax.scatter(data3_pca[:, 0], data3_pca[:, 1], alpha=0.5, label='DP-ARF Data')
    ax.legend()
    ax.set_title('PCA - All Data Overlap')

    plt.tight_layout()
    plt.show()

def calculate_additional_metrics(original_data, og_arf_data, dp_arf_data):
    metrics = {}

    original_data_sampled = original_data.sample(n=og_arf_data.shape[0])

    # Calculate Wasserstein Distance for numeric columns
    for col in original_data.select_dtypes(include=['int64', 'float64']).columns:
        wasserstein_og = wasserstein_distance(original_data_sampled[col], og_arf_data[col])
        wasserstein_dp = wasserstein_distance(original_data_sampled[col], dp_arf_data[col])
        metrics[f'Wasserstein Distance {col}'] = {'OG ARF': wasserstein_og, 'DP ARF': wasserstein_dp}

    # Calculate Chi-Square Test for categorical columns
    for col in original_data.select_dtypes(include=['category', 'object']).columns:
        original_counts = original_data_sampled[col].value_counts().sort_index()
        og_arf_counts = og_arf_data[col].value_counts().sort_index()
        dp_arf_counts = dp_arf_data[col].value_counts().sort_index()

        categories = original_counts.index.union(og_arf_counts.index).union(dp_arf_counts.index)
        original_counts = original_counts.reindex(categories, fill_value=0)
        og_arf_counts = og_arf_counts.reindex(categories, fill_value=0)
        dp_arf_counts = dp_arf_counts.reindex(categories, fill_value=0)

        chi2_stat_og, chi2_pvalue_og = chisquare(original_counts, og_arf_counts)
        chi2_stat_dp, chi2_pvalue_dp = chisquare(original_counts, dp_arf_counts)
        metrics[f'Chi-Square Test {col}'] = {'OG ARF': chi2_stat_og, 'DP ARF': chi2_stat_dp}

    # Calculate RMSE and MAE for numeric columns
```

```
    for col in original_data.select_dtypes(include=['int64', 'float64']).columns:
        rmse_og = np.sqrt(mean_squared_error(original_data_sampled[col], og_arf_data[col]))
        rmse_dp = np.sqrt(mean_squared_error(original_data_sampled[col], dp_arf_data[col]))
        mae_og = np.mean(np.abs(original_data_sampled[col] - og_arf_data[col]))
        mae_dp = np.mean(np.abs(original_data_sampled[col] - dp_arf_data[col]))
        metrics[f'RMSE {col}'] = {'OG ARF': rmse_og, 'DP ARF': rmse_dp}
        metrics[f'MAE {col}'] = {'OG ARF': mae_og, 'DP ARF': mae_dp}

    return metrics

# Calculate correlation matrices for datasets
def get_correlation_matrices(data1, data2, data3):
    corr_matrices = {
        "Original Data": data1.corr(),
        "OG ARF Data": data2.corr(),
        "DP-ARF Data": data3.corr()
    }
    return corr_matrices

# Evaluate classification model on synthetic data fitted on original data
def model_evaluation_on_classification(original_data, synthetic_data, target_col='Medical Condition'):
    og = original_data.copy()
    dp = synthetic_data.copy()
    og[target_col] = (og['Medical Condition'] == 'Cancer').astype(int)
    dp[target_col] = (dp['Medical Condition'] == 'Cancer').astype(int)

    X_orig = og.drop(columns=[target_col])
    y_orig = og[target_col]

    X_synth = dp.drop(columns=[target_col])
    y_synth = dp[target_col]

    categorical_features = X_orig.select_dtypes(include=['object', 'category']).columns
    numeric_features = X_orig.select_dtypes(include=[np.number]).columns

    preprocessor = ColumnTransformer(
        transformers=[
            ('num', 'passthrough', numeric_features),
            ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
        ])

    X_encoded_orig = preprocessor.fit_transform(X_orig)
    X_encoded_synth = preprocessor.transform(X_synth)

    clf = LogisticRegression(max_iter=2000)
```

```
178     clf.fit(X_encoded_orig, y_orig)
179 
180     return clf.score(X_encoded_synth, y_synth)
181 
182 # Preprocessing function to encode categorical variables
183 def preprocess_data(df):
184     categorical_features = df.select_dtypes(include=['category']).columns
185     numeric_features = df.select_dtypes(include=[np.number]).columns
186 
187     preprocessor = ColumnTransformer(
188         transformers=[
189             ('num', 'passthrough', numeric_features),
190             ('cat', OneHotEncoder(handle_unknown='ignore'),
    categorical_features)
191         ])
192 
193     return preprocessor.fit_transform(df), preprocessor
194 
195 # Reconstruction Attack
196 def reconstruction_attack(original_data, synthetic_data):
197     X_orig, orig_preprocessor = preprocess_data(original_data.drop(
    columns=['Billing Amount']))
198     X_synth, synth_preprocessor = preprocess_data(synthetic_data.drop(
    columns=['Billing Amount']))
199 
200     nn = NearestNeighbors(n_neighbors=3)
201     nn.fit(X_synth)
202     distances, indices = nn.kneighbors(X_orig)
203 
204     return distances.mean()
205 
206 # Post process synthetic data to ensure valid values
207 def post_process_synthetic_data(data):
208     data.loc[data['Room Number'] < 0, 'Room Number'] = data['Room Number'
    ].mean()
209     data.loc[data['Age'] < 0, 'Age'] = data['Age'].mean()
210     data['Age'] = data['Age'].astype('int')
211     data['Room Number'] = data['Room Number'].astype('int')
212     mean_billing = data['Billing Amount'][data['Billing Amount'] >= 0].
    mean()
213     data.loc[data['Billing Amount'] < 0, 'Billing Amount'] = mean_billing
214     data['Billing Amount'] = data['Billing Amount'].round(2)
215     return data
216 
217 
218 # Main Script
219 file_path = '..\\datasets\\healthcare_dataset.csv'
220 data = load_new_dataset(file_path)
221 
222 # Generate synthetic data using OG ARF model
223 arf_model = OG_ARF.arf(data)
224 arf_model.forde()
225 og_arf_data = arf_model.forge(data.shape[0])
```

```
og_arf_data = post_process_synthetic_data(og_arf_data)

# Generate synthetic data using DP-ARF model
dp_arf_model = DP_ARF(data, epsilon=0.1)
dp_arf_model.forde()
dp_arf_data = dp_arf_model.forge(data.shape[0])
dp_arf_data = post_process_synthetic_data(dp_arf_data)

# Print the first few rows of the datasets
print('original data: ', data.head())
print('OG arf data: ', og_arf_data.head())
print('dp arf data: ', dp_arf_data.head())

# Identify numeric and categorical columns
numeric_columns = data.select_dtypes(include=[np.number]).columns.tolist()
categorical_columns = data.select_dtypes(include=['category', 'object']).columns.tolist()

print("Numeric columns:", numeric_columns)
print("Categorical columns:", categorical_columns)

# Plot density distributions
plot_density_distribution(data, og_arf_data, dp_arf_data, numeric_columns, sample_size=100)

numeric_data = data.select_dtypes(include=[np.number])
numeric_og_arf_data = og_arf_data.select_dtypes(include=[np.number])
numeric_dp_arf_data = dp_arf_data.select_dtypes(include=[np.number])

# Plot PCA results
plot_pca(numeric_data, numeric_og_arf_data, numeric_dp_arf_data, sample_size=100)

# Calculate and print correlation matrices
correlation_matrices = get_correlation_matrices(numeric_data, numeric_og_arf_data, numeric_dp_arf_data)
for key, value in correlation_matrices.items():
    print(f'{key} Correlation Matrix:')
    print(value)

# Calculate and print additional metrics
additional_metrics = calculate_additional_metrics(data, og_arf_data, dp_arf_data)
for metric, values in additional_metrics.items():
    print(f'{metric}:')
    for method, value in values.items():
        print(f'  {method}: {value}')

# Evaluate classification accuracy
accuracy_og = model_evaluation_on_classification(data, og_arf_data)
accuracy_dp = model_evaluation_on_classification(data, dp_arf_data)
print(f'Classification Accuracy (OG ARF): {accuracy_og}')
```

```
273 print(f'Classification Accuracy (DP ARF): {accuracy_dp}')
274
275 # Evaluate Attacks
276
277 # Evaluate reconstruction attack on og arf data
278 reconstruction_error = reconstruction_attack(data, og_arf_data)
279 print(f'Reconstruction Attack Error OG ARF (Mean Distance): {
        reconstruction_error}')
280
281 # Evaluate reconstruction attack on dp arf data
282 reconstruction_error = reconstruction_attack(data, dp_arf_data)
283 print(f'Reconstruction Attack Error DP ARF (Mean Distance): {
        reconstruction_error}')
```

Appendix A.5: Evaluation Script for Adult Dataset

```
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error, accuracy_score, roc_auc_score, precision_score, recall_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.neighbors import NearestNeighbors
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from scipy.stats import chisquare, wasserstein_distance
import seaborn as sns
import sys
import os

# Add the parent directory to the system path
sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath('dp_arf.py'))))
from dp_arf import arf as DP_ARF  # Import the DP_ARF class
from arfpy import arf as OG_ARF  # Import the OG ARF class without differential privacy

def load_uci_adult_dataset():
    column_names = [
        'age', 'workclass', 'fnlwgt', 'education', 'education_num',
        'marital_status', 'occupation', 'relationship', 'race', 'sex',
        'capital_gain', 'capital_loss', 'hours_per_week', 'native_country', 'income'
    ]
    url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data'
    data = pd.read_csv(url, header=None, names=column_names, na_values='?', skipinitialspace=True)

    return data

def plot_density_distribution(data1, data2, data3, columns, sample_size=1000):
    # Sample the data if it exceeds the sample size
    if data1.shape[0] > sample_size:
        data1_sample = data1.sample(n=sample_size)
        data2_sample = data2.sample(n=sample_size)
        data3_sample = data3.sample(n=sample_size)
    else:
        data1_sample = data1
        data2_sample = data2
```

```
        data3_sample = data3

    # Initialize plot count
    plot_count = 0

    # Create subplots iteratively
    for i in range(0, len(columns), 4):
        num_cols = min(4, len(columns) - i)
        if num_cols == 1:
            fig, axs = plt.subplots(1, 1, figsize=(7, 5))
            axs = [axs]
        elif num_cols == 2:
            fig, axs = plt.subplots(1, 2, figsize=(14, 5))
        elif num_cols == 3:
            fig, axs = plt.subplots(2, 2, figsize=(14, 10))
            axs[1, 1].remove()  # Remove the unused subplot
            axs = [axs[0, 0], axs[0, 1], axs[1, 0]]
        else:  # num_cols == 4
            fig, axs = plt.subplots(2, 2, figsize=(14, 10))
            axs = axs.flatten()  # Flatten the 2D array of axes to easily
     index them

        for j, column in enumerate(columns[i:i+num_cols]):
            if pd.api.types.is_numeric_dtype(data1_sample[column]):
                # Plot the density distributions on the corresponding
    subplot
                data1_sample[column].plot(kind='density', ax=axs[j],
    label='Original Data', legend=True)
                data2_sample[column].plot(kind='density', ax=axs[j],
    label='OG ARF Data', legend=True)
                data3_sample[column].plot(kind='density', ax=axs[j],
    label='DP-ARF Data', legend=True)
                axs[j].set_title(f'Density Distribution for {column}')
                axs[j].legend()
            else:
                axs[j].set_title(f'Skipping column {column} as it is not
    numeric.')
                axs[j].axis('off')

        # Adjust layout
        plt.tight_layout()
        plt.show()

def plot_pca(data1, data2, data3, sample_size=1000):
    if data1.shape[0] > sample_size:
        data1_sample = data1.sample(n=sample_size)
        data2_sample = data2.sample(n=sample_size)
        data3_sample = data3.sample(n=sample_size)
    else:
        data1_sample = data1
        data2_sample = data2
        data3_sample = data3

```

```
    # Select only numeric columns for PCA
    data1_numeric = data1_sample.select_dtypes(include=[np.number])
    data2_numeric = data2_sample.select_dtypes(include=[np.number])
    data3_numeric = data3_sample.select_dtypes(include=[np.number])

    # Perform PCA
    pca = PCA(n_components=2)
    data1_pca = pca.fit_transform(data1_numeric)
    data2_pca = pca.transform(data2_numeric)
    data3_pca = pca.transform(data3_numeric)

    fig, ax = plt.subplots(figsize=(10, 8))  # Adjusted grid size

    ax.scatter(data1_pca[:, 0], data1_pca[:, 1], alpha=0.5, label='Original Data')
    ax.scatter(data2_pca[:, 0], data2_pca[:, 1], alpha=0.5, label='OG ARF Data')
    ax.scatter(data3_pca[:, 0], data3_pca[:, 1], alpha=0.5, label='DP-ARF Data')
    ax.legend()
    ax.set_title('PCA - All Data Overlap')

    plt.tight_layout()
    plt.show()

def calculate_additional_metrics(original_data, og_arf_data, dp_arf_data):
    metrics = {}

    original_data_sampled = original_data.sample(n=og_arf_data.shape[0])

    # Calculate Wasserstein Distance for numeric columns
    for col in original_data.select_dtypes(include=['int64', 'float64']).columns:
        wasserstein_og = wasserstein_distance(original_data_sampled[col], og_arf_data[col])
        wasserstein_dp = wasserstein_distance(original_data_sampled[col], dp_arf_data[col])
        metrics[f'Wasserstein Distance {col}'] = {'OG ARF': wasserstein_og, 'DP ARF': wasserstein_dp}

    # Calculate Chi-Square Test for categorical columns
    for col in original_data.select_dtypes(include=['category', 'object']).columns:
        original_counts = original_data_sampled[col].value_counts().sort_index()
        og_arf_counts = og_arf_data[col].value_counts().sort_index()
        dp_arf_counts = dp_arf_data[col].value_counts().sort_index()

        categories = original_counts.index.union(og_arf_counts.index).union(dp_arf_counts.index)
        original_counts = original_counts.reindex(categories, fill_value=0)
```

```
133         og_arf_counts = og_arf_counts.reindex(categories, fill_value=0)
134         dp_arf_counts = dp_arf_counts.reindex(categories, fill_value=0)
135 
136         chi2_stat_og, chi2_pvalue_og = chisquare(original_counts,
    og_arf_counts)
137         chi2_stat_dp, chi2_pvalue_dp = chisquare(original_counts,
    dp_arf_counts)
138         metrics[f'Chi-Square Test {col}'] = {'OG ARF': chi2_stat_og, 'DP
    ARF': chi2_stat_dp}
139 
140     # Calculate RMSE and MAE for numeric columns
141     for col in original_data.select_dtypes(include=['int64', 'float64']).
    columns:
142         rmse_og = np.sqrt(mean_squared_error(original_data_sampled[col],
    og_arf_data[col]))
143         rmse_dp = np.sqrt(mean_squared_error(original_data_sampled[col],
    dp_arf_data[col]))
144         mae_og = np.mean(np.abs(original_data_sampled[col] - og_arf_data[
    col]))
145         mae_dp = np.mean(np.abs(original_data_sampled[col] - dp_arf_data[
    col]))
146         metrics[f'RMSE {col}'] = {'OG ARF': rmse_og, 'DP ARF': rmse_dp}
147         metrics[f'MAE {col}'] = {'OG ARF': mae_og, 'DP ARF': mae_dp}
148 
149     return metrics
150 
151 # Calculate correlation matrices for datasets
152 def get_correlation_matrices(data1, data2, data3):
153     corr_matrices = {
154         "Original Data": data1.corr(),
155         "OG ARF Data": data2.corr(),
156         "DP-ARF Data": data3.corr()
157     }
158     return corr_matrices
159 
160 # Evaluate classification model on synthetic data fitted on original data
161 def model_evaluation_on_classification(original_data, synthetic_data,
    target_col='income'):
162     og = original_data.copy()
163     dp = synthetic_data.copy()
164     og[target_col] = (og['income'] == '>50K').astype(int)
165     dp[target_col] = (dp['income'] == '>50K').astype(int)
166 
167     X_orig = og.drop(columns=[target_col])
168     y_orig = og[target_col]
169 
170     X_synth = dp.drop(columns=[target_col])
171     y_synth = dp[target_col]
172 
173     categorical_features = X_orig.select_dtypes(include=['object', '
    category']).columns
174     numeric_features = X_orig.select_dtypes(include=[np.number]).columns
175 
```

```
    preprocessor = ColumnTransformer(
        transformers=[
            ('num', 'passthrough', numeric_features),
            ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
        ])

    X_encoded_orig = preprocessor.fit_transform(X_orig)
    X_encoded_synth = preprocessor.transform(X_synth)

    clf = LogisticRegression(max_iter=2000)
    clf.fit(X_encoded_orig, y_orig)

    return clf.score(X_encoded_synth, y_synth)

# Preprocessing function to encode categorical variables
def preprocess_data(df):
    categorical_features = df.select_dtypes(include=['category']).columns
    numeric_features = df.select_dtypes(include=[np.number]).columns

    preprocessor = ColumnTransformer(
        transformers=[
            ('num', 'passthrough', numeric_features),
            ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
        ])

    return preprocessor.fit_transform(df), preprocessor

# Reconstruction Attack
def reconstruction_attack(original_data, synthetic_data):
    X_orig, orig_preprocessor = preprocess_data(original_data.drop(columns=['income']))
    X_synth, synth_preprocessor = preprocess_data(synthetic_data.drop(columns=['income']))

    nn = NearestNeighbors(n_neighbors=3)
    nn.fit(X_synth)
    distances, indices = nn.kneighbors(X_orig)

    return distances.mean()

# Post process synthetic data to ensure valid values
def post_process_synthetic_data(data):
    data.loc[data['age'] < 0, 'age'] = data['age'].mean()
    data['age'] = data['age'].astype('int')

    mean_fnlwgt = data['fnlwgt'][data['fnlwgt'] >= 0].mean()
    data.loc[data['fnlwgt'] < 0, 'fnlwgt'] = mean_fnlwgt
    data['fnlwgt'] = data['fnlwgt'].astype('int')

    mean_education_num = data['education_num'][data['education_num'] >= 0].mean()
```

```
    data.loc[data['education_num'] < 0, 'education_num'] =
    mean_education_num
    data['education_num'] = data['education_num'].astype('int')

    mean_capital_gain = data['capital_gain'][data['capital_gain'] >= 0].
    mean()
    data.loc[data['capital_gain'] < 0, 'capital_gain'] =
    mean_capital_gain
    data['capital_gain'] = data['capital_gain'].astype('int')

    mean_capital_loss = data['capital_loss'][data['capital_loss'] >= 0].
    mean()
    data.loc[data['capital_loss'] < 0, 'capital_loss'] =
    mean_capital_loss
    data['capital_loss'] = data['capital_loss'].astype('int')

    mean_hours_per_week = data['hours_per_week'][data['hours_per_week']
    >= 0].mean()
    data.loc[data['hours_per_week'] < 0, 'hours_per_week'] =
    mean_hours_per_week
    data['hours_per_week'] = data['hours_per_week'].astype('int')

    return data


# Main Script
data = load_uci_adult_dataset()

# Generate synthetic data using OG ARF model
arf_model = OG_ARF.arf(data)
arf_model.forde()
og_arf_data = arf_model.forge(data.shape[0])
og_arf_data = post_process_synthetic_data(og_arf_data)

# Generate synthetic data using DP-ARF model
dp_arf_model = DP_ARF(data, epsilon=0.5)
dp_arf_model.forde()
dp_arf_data = dp_arf_model.forge(data.shape[0])
dp_arf_data = post_process_synthetic_data(dp_arf_data)

# Print the first few rows of the datasets
print('original data: ', data.head())
print('OG arf data: ', og_arf_data.head())
print('dp arf data: ', dp_arf_data.head())

# Identify numeric and categorical columns
numeric_columns = data.select_dtypes(include=[np.number]).columns.tolist
    ()
categorical_columns = data.select_dtypes(include=['category', 'object']).
    columns.tolist()

print("Numeric columns:", numeric_columns)
print("Categorical columns:", categorical_columns)
```

```
268
269 # Plot density distributions
270 plot_density_distribution(data, og_arf_data, dp_arf_data, numeric_columns
        , sample_size=100)
271
272 # Plot PCA results
273 numeric_data = data.select_dtypes(include=[np.number])
274 numeric_og_arf_data = og_arf_data.select_dtypes(include=[np.number])
275 numeric_dp_arf_data = dp_arf_data.select_dtypes(include=[np.number])
276
277 plot_pca(numeric_data, numeric_og_arf_data, numeric_dp_arf_data,
        sample_size=100)
278
279 # Calculate and print correlation matrices
280 correlation_matrices = get_correlation_matrices(numeric_data,
        numeric_og_arf_data, numeric_dp_arf_data)
281 for key, value in correlation_matrices.items():
282     print(f'{key} Correlation Matrix:')
283     print(value)
284
285 # Calculate and print additional metrics
286 additional_metrics = calculate_additional_metrics(data, og_arf_data,
        dp_arf_data)
287 for metric, values in additional_metrics.items():
288     print(f'{metric}:')
289     for method, value in values.items():
290         print(f'  {method}: {value}')
291
292 # Evaluate classification accuracy
293 accuracy_og = model_evaluation_on_classification(data, og_arf_data)
294 accuracy_dp = model_evaluation_on_classification(data, dp_arf_data)
295 print(f'Classification Accuracy (OG ARF): {accuracy_og}')
296 print(f'Classification Accuracy (DP ARF): {accuracy_dp}')
297
298 # Evaluate Attacks
299
300 # Evaluate reconstruction attack on og arf data
301 reconstruction_error = reconstruction_attack(data, og_arf_data)
302 print(f'Reconstruction Attack Error OG ARF (Mean Distance): {
        reconstruction_error}')
303
304 # Evaluate reconstruction attack on dp arf data
305 reconstruction_error = reconstruction_attack(data, dp_arf_data)
306 print(f'Reconstruction Attack Error DP ARF (Mean Distance): {
        reconstruction_error}')
```

Appendix B: Additional Results

Appendix B.1: Extended Results Plots

This subsection includes additional plots generated during the evaluation phase, that are not included in the main body of the report.

For healthcare epsilon = 1

Name	Age	Gender	Blood Type	Medical Condition
Bobby Jackson	30	Male	B-	Cancer
Leslie Terry	62	Male	A+	Obesity
Danny Smith	76	Female	A-	Obesity
Andrew Watts	28	Female	O+	Diabetes
Adrienne Bell	43	Female	AB+	Cancer

Table 70: Original Healthcare Data Part 1 $\epsilon = 1$

Billing Amount	Room Number	Medication	Test Results
18856.28	328	Paracetamol	Normal
33643.33	265	Ibuprofen	Inconclusive
27955.10	205	Aspirin	Normal
37909.78	450	Ibuprofen	Abnormal
14238.32	458	Penicillin	Abnormal

Table 71: Original Healthcare Data Part 2 $\epsilon = 1$

Name	Age	Gender	Blood Type	Medical Condition
Jamie Hammond	27	Male	AB+	Obesity
David Wright	33	Male	B-	Diabetes
Jesse Sanchez	29	Female	A-	Hypertension
Sandy Camacho	60	Female	O-	Cancer
Manuel Barber	56	Male	A+	Asthma

Table 72: OG ARF Healthcare Data Part 1 $\epsilon = 1$

Billing Amount	Room Number	Medication	Test Results
22586.63	362	Aspirin	Inconclusive
39095.18	236	Paracetamol	Abnormal
11122.12	550	Paracetamol	Inconclusive
11836.36	175	Penicillin	Normal
5696.55	481	Penicillin	Abnormal

Table 73: OG ARF Healthcare Data Part 2 $\epsilon = 1$

Name	Age	Gender	Blood Type	Medical Condition
Kenneth Leonard	57	Male	A+	Hypertension
Kevin Williams	106	Male	O-	Arthritis
Christina Gentry	44	Male	AB-	Arthritis
Danielle Daniels	52	Female	B-	Diabetes
Brooke Burns	82	Male	AB-	Cancer

Table 74: DP ARF Healthcare Data Part 1 $\epsilon = 1$

Billing Amount	Room Number	Medication	Test Results
31822.23	390	Aspirin	Normal
34732.82	352	Aspirin	Abnormal
25458.48	245	Lipitor	Inconclusive
36739.01	241	Lipitor	Abnormal
6654.31	237	Aspirin	Inconclusive

Table 75: DP ARF Healthcare Data Part 2 $\epsilon = 1$

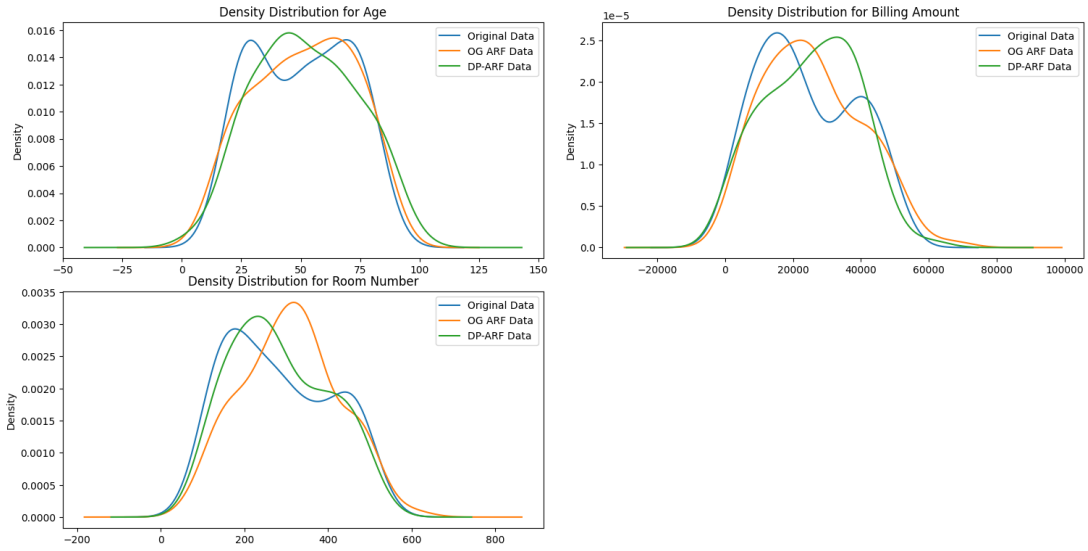

Figure 14: Density distribution for Healthcare $\epsilon = 1$

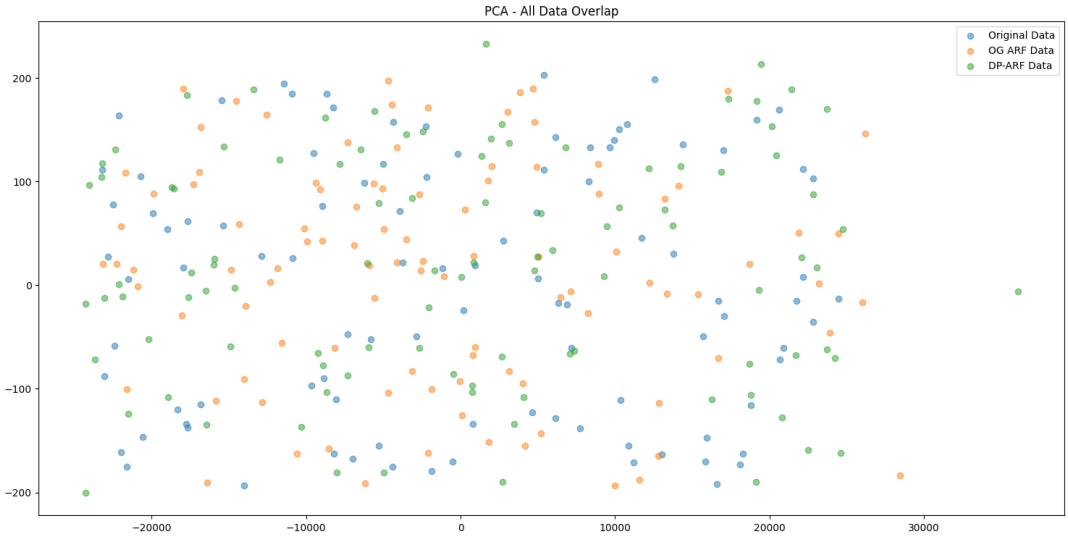

Figure 15: PCA analysis for Healthcare $\epsilon = 1$

Features	Original Data	OG ARF Data	DP ARF Data
Age vs Billing Amount	-0.0038	-0.0064	-0.0019
Age vs Room Number	-0.0007	-0.0034	0.0027
Billing Amount vs Room Number	-0.0029	-0.0024	-0.0051

Table 76: Correlation Matrices for Healthcare $\epsilon = 1$

Feature	OG ARF	DP ARF
Age	1.2608	1.2965
Billing Amount	1149.3753	1099.1245
Room Number	6.6086	5.3032

Table 77: Wasserstein Distance for Healthcare $\epsilon = 1$

Feature	OG ARF	DP ARF
Age	27.5540	27.8962
Billing Amount	19564.2468	19789.1707
Room Number	161.8494	163.3155

Table 78: RMSE for Healthcare $\epsilon = 1$

Feature	OG ARF	DP ARF
Age	22.4777	22.6945
Billing Amount	15948.3541	16147.0625
Room Number	131.76081	132.8233

Table 79: MAE for Healthcare $\epsilon = 1$

ARF	Value
OG	0.8253
DP	0.8245

Table 80: Classification Accuracy for Healthcare $\epsilon = 1$

ARF	Value
OG	0.4428
DP	0.4515

Table 81: Reconstruction Attack Error for Healthcare $\epsilon = 1$

For adult epsilon = 0.1

Age	Workclass	Education	Occupation
39	State-gov	Bachelors	Adm-clerical
50	Self-emp-not-inc	Bachelors	Exec-managerial
38	Private	HS-grad	Handlers-cleaners
53	Private	11th	Handlers-cleaners
28	Private	Bachelors	Prof-specialty

Table 82: Original Data for Adult $\epsilon = 0.1$

Age	Workclass	Education	Occupation
74	?	Masters	?
49	Private	Assoc-acdm	Tech-support
39	Private	Some-college	Exec-managerial
47	Local-gov	10th	Protective-serv
41	Private	HS-grad	Craft-repair

Table 83: OG ARF Data for Adult $\epsilon = 0.1$

Age	Workclass	Education	Occupation
48	State-gov	Masters	Adm-clerical
47	Federal-gov	HS-grad	Adm-clerical
55	Private	Some-college	Exec-managerial
48	Self-emp-not-inc	Bachelors	Exec-managerial
39	Private	HS-grad	Sales

Table 84: DP ARF Data for Adult $\epsilon = 0.1$

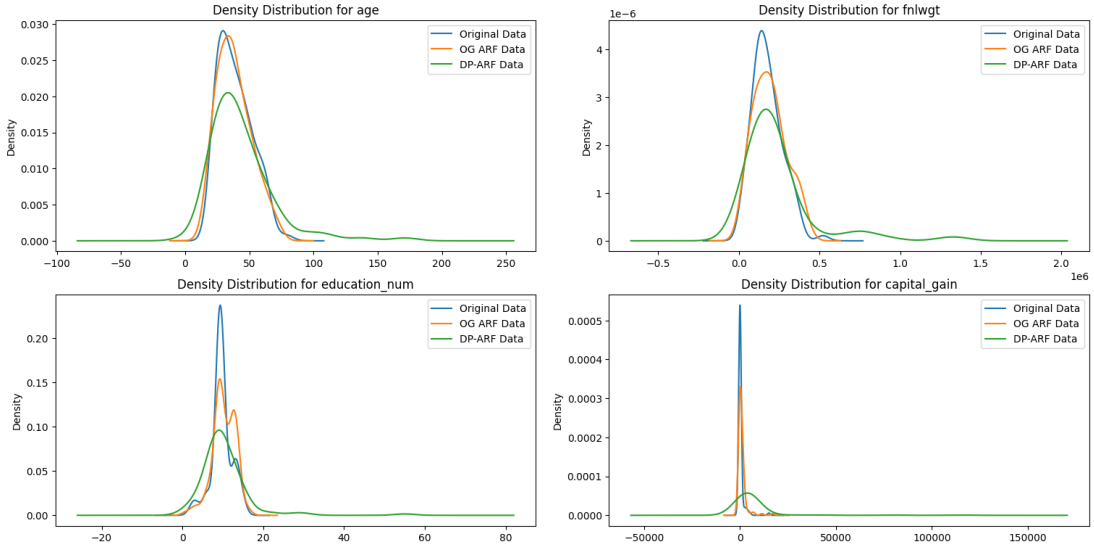

Figure 16: Density distribution for Adult Part 1 $\epsilon = 0.1$

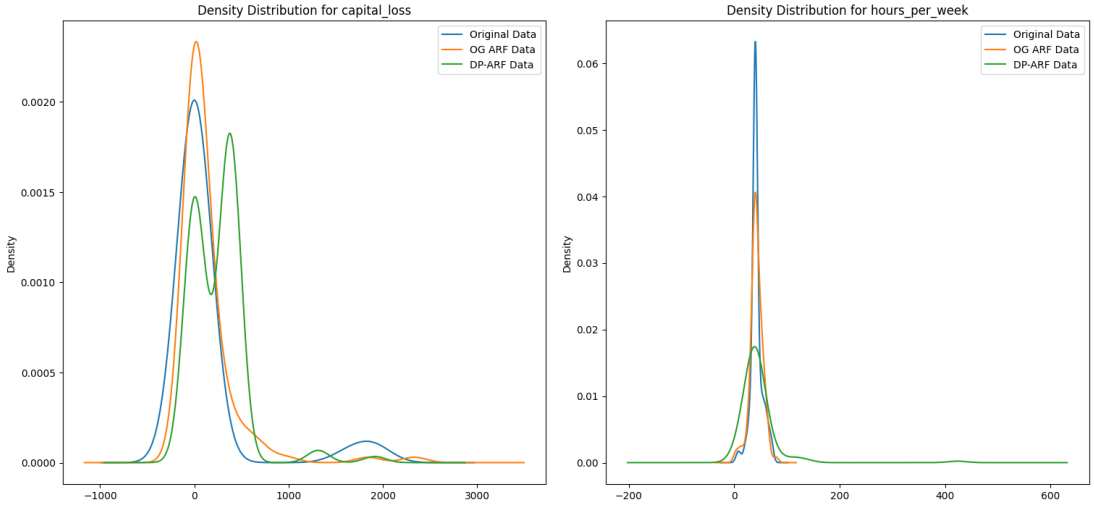

Figure 17: Density distribution for Adult Part 2 $\epsilon = 0.1$

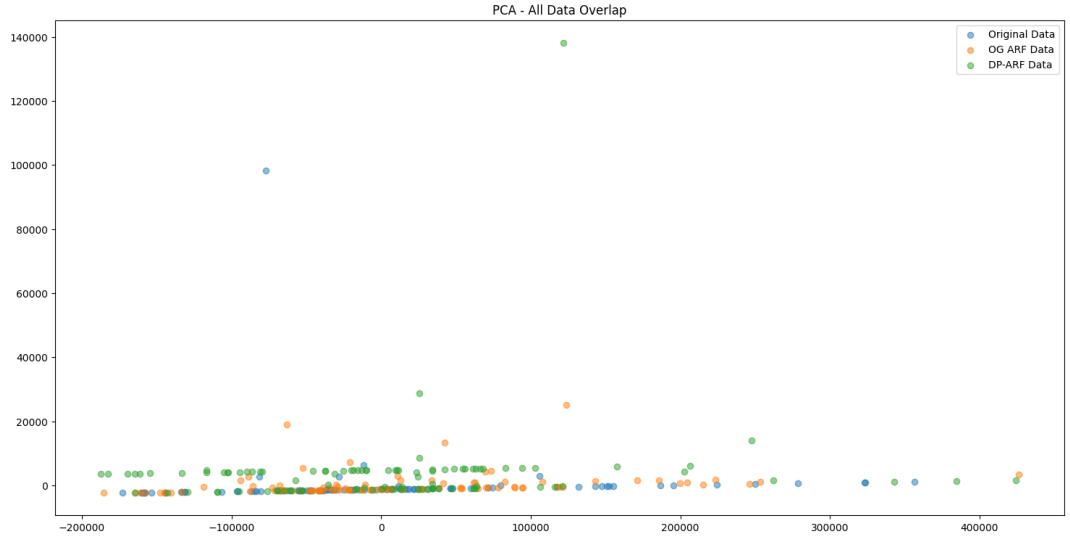

Figure 18: PCA analysis for Adult $\epsilon = 0.1$

Features	Original Data	OG ARF Data	DP ARF Data
Age	1.0000	1.0000	1.0000
Fnlwgt	-0.0766	-0.0721	0.0059
Education Num	0.0365	0.0194	0.0413
Capital Gain	0.0777	0.0972	0.0504
Capital Loss	0.0578	0.0625	0.0348
Hours per Week	0.0688	0.0768	0.0355

Table 85: Correlation Matrices for Adult $\epsilon = 0.1$

Feature	OG ARF	DP ARF
Age	0.2214	3.7907
Fnlwgt	5414.3372	37159.7450
Education Num	0.3016	0.9178
Capital Gain	749.7283	4935.6195
Capital Loss	69.9394	292.7822
Hours per Week	1.5807	6.3299

Table 86: Wasserstein Distances for Adult $\epsilon = 0.1$

Feature	OG ARF	DP ARF
Workclass	3.8369	7911.3941
Education	18.7182	1615.2482
Marital Status	4.3219	1885.0502
Occupation	12.6491	1309.0290
Relationship	4.4634	1286.4095
Race	3.5649	3962.1131
Sex	1.9867	41.5534
Native Country	∞	6387.2342
Income	0.1619	624.3457

Table 87: Chi-Square Tests for Adult $\epsilon = 0.1$

Feature	OG ARF	DP ARF
Age	19.3568	29.6035
Fnlwgt	149636.8829	249691.3420
Education Num	3.6857	5.4593
Capital Gain	10300.8001	25770.0017
Capital Loss	553.3943	1041.7032
Hours per Week	17.4520	30.8195

Table 88: RMSE for Adult $\epsilon = 0.1$

Feature	OG ARF	DP ARF
Age	15.4966	18.7283
Fnlwgt	114600.2252	140842.3979
Education Num	2.8013	3.4639
Capital Gain	2565.5560	6585.2397
Capital Loss	208.3861	436.7987
Hours per Week	12.9424	17.4454

Table 89: MAE for Adult $\epsilon = 0.1$

ARF	Value
OG	0.8240
DP	0.6686

Table 90: Classification Accuracy for Adult $\epsilon = 0.1$

ARF	Value
OG	250.1662
DP	269.4452

Table 91: Reconstruction Attack Error for Adult $\epsilon = 0.1$

For adult epsilon = 1

Age	Workclass	Education	Marital Status
39	State-gov	Bachelors	Never-married
50	Self-emp-not-inc	Bachelors	Married-civ-spouse
38	Private	HS-grad	Divorced
53	Private	11th	Married-civ-spouse
28	Private	Bachelors	Married-civ-spouse

Table 92: Original Data Sample for Adult $\epsilon = 1.0$

Age	Workclass	Education	Marital Status
21	Private	HS-grad	Married-civ-spouse
46	Private	Assoc	Married-civ-spouse
27	?	Some-college	Married-civ-spouse
75	?	Some-college	Married-civ-spouse
42	State-gov	HS-grad	Married-civ-spouse

Table 93: OG ARF Data Sample for Adult $\epsilon = 1.0$

Age	Workclass	Education	Marital Status
24	Private	Masters	Divorced
27	Local-gov	HS-grad	Never-married
45	Private	Assoc	Married-civ-spouse
21	Private	Some-college	Never-married
56	Self-emp-not-inc	HS-grad	Married-civ-spouse

Table 94: DP ARF Data Sample for Adult $\epsilon = 1.0$

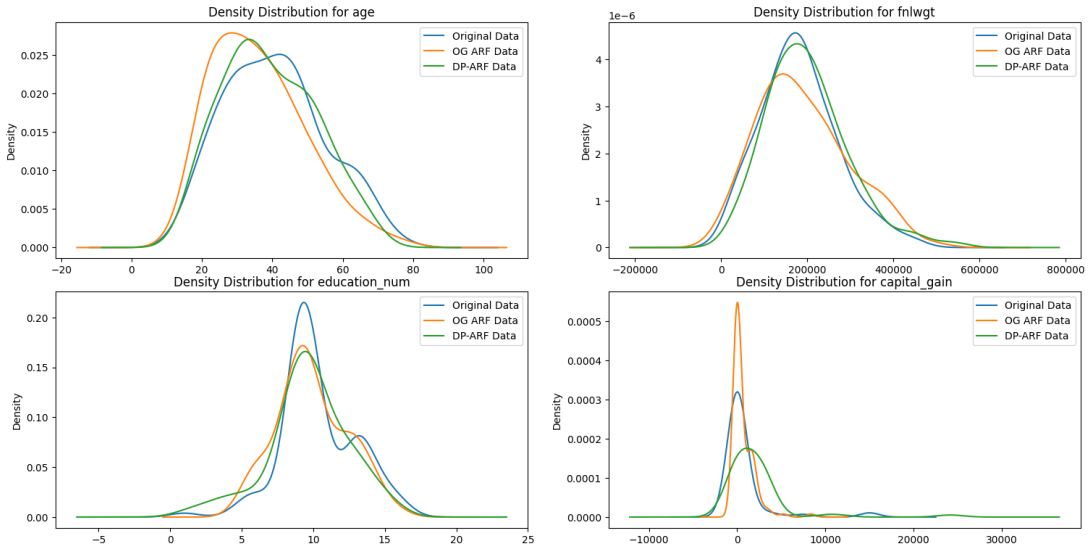

Figure 19: Density distribution for Adult Part 1 $\epsilon = 1$

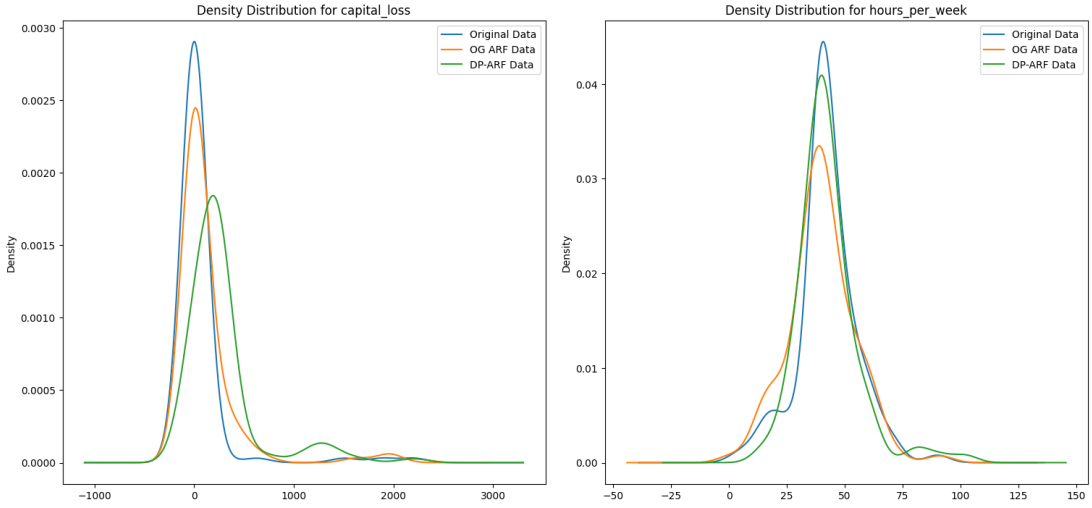

Figure 20: Density distribution for Adult Part 2 $\epsilon = 1$

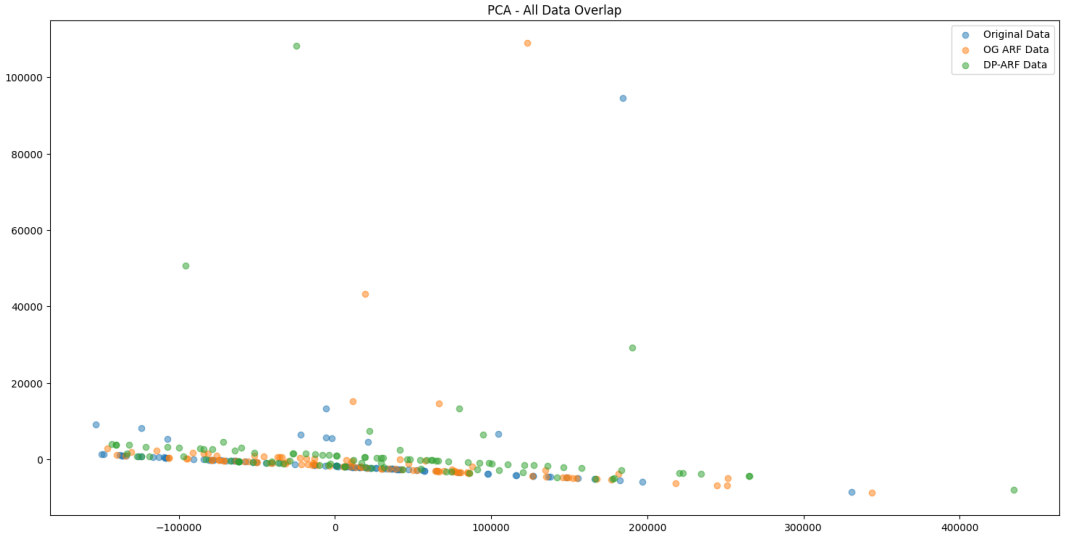

Figure 21: PCA analysis for Adult $\epsilon = 1$

Features	Original Data	OG ARF Data	DP ARF Data
Age	1.0000	1.0000	1.0000
Fnlwgt	-0.0766	-0.0697	-0.0677
Education Num	0.0365	0.0245	0.0403
Capital Gain	0.0777	0.1020	0.0915
Capital Loss	0.0578	0.0648	0.0684
Hours per Week	0.0688	0.0765	0.0589

Table 95: Correlation Matrices for Adult $\epsilon = 1.0$

Feature	OG ARF	DP ARF
Age	0.3221	0.3377
Fnlwgt	4851.0239	6277.4043
Education Num	0.3251	0.4910
Capital Gain	740.1327	1826.7626
Capital Loss	86.0784	163.6468
Hours per Week	1.5608	1.6994

Table 96: Wasserstein Distances for Adult $\epsilon = 1.0$

Feature	OG ARF	DP ARF
Workclass	11.9714	4751.8106
Education	13.1787	1359.2558
Marital Status	3.6137	1698.6054
Occupation	11.2638	961.6070
Relationship	5.4163	1291.7518
Race	5.1088	2822.8779
Sex	0.5349	115.6228
Native Country	53.9169	4331.7595
Income	0.0284	312.0827

Table 97: Chi-Square Tests for Adult $\epsilon = 1.0$

Feature	OG ARF	DP ARF
Age	19.2654	19.4251
Fnlwgt	151081.7418	151746.2248
Education Num	3.6755	3.7139
Capital Gain	10537.4007	10968.1428
Capital Loss	550.0625	564.7148
Hours per Week	17.5370	17.7166

Table 98: RMSE for Adult $\epsilon = 1.0$

Feature	OG ARF	DP ARF
Age	15.3875	15.4439
Fnlwgt	113848.5477	114413.2851
Education Num	2.8206	2.8668
Capital Gain	2577.9933	3558.1125
Capital Loss	219.2349	296.3991
Hours per Week	13.0036	13.0327

Table 99: MAE for Adult $\epsilon = 1.0$

ARF	Value
OG	0.8268
DP	0.7469

Table 100: Classification Accuracy for Adult $\epsilon = 1.0$

ARF	Value
OG	248.3583
DP	311.3938

Table 101: Reconstruction Attack Error for Adult $\epsilon = 1.0$

Appendix C: User Manual for running the scripts

```
By following the below directory structure after downloading the
supplementary materials and ensuring the given packages are installed
you can run the test scripts.

Note: to change the epsilon value edit the epsilon parameter in the
    DP_ARF constructor of the test scripts.

## Directory Structure

Supplementary material/
          dp_arf.py
          datasets/
              <dataset_files>
          testscripts/
              adult.py
              health.py
              insurance.py


- `datasets/`: Directory containing the dataset files.
- `dp_arf.py`: Python file containing the ARF class with implemented
    Differential Privacy.
- `testscripts/`: Directory containing the scripts to generate and
    evaluate synthetic data.

## Prerequisites

Ensure you have the following packages installed:

- numpy
- pandas
- matplotlib
- seaborn
- scikit-learn
- scipy
- arfpy

You can install these packages using pip:
`pip install numpy pandas matplotlib seaborn scikit-learn scipy arfpy`


## Running the Scripts

```

```
1. Adult Dataset Script
To run the script for the UCI Adult dataset:
‘python testscripts/adult.py‘

2. Health Dataset Script
To run the script for the Healthcare dataset:
‘python testscripts/health.py‘

3. Insurance Dataset Script
To run the script for the Insurance dataset:
‘python testscripts/insurance.py‘
```