Student Numbers: 73129, 72997

1. Data De-Identification

The first phase of the project involved implementing a simple de-identification process on the med_data table. Our approach was to create a new, separate table named med_data_di (using the MedDataDeidentified Java class).

This process involved the removal (suppression) of all direct identifiers that could uniquely identify an individual. The fields removed were name, address and email.

The resulting med_data_di table retained only the quasi-identifiers (QIs) (age, gender, postal_code) and the sensitive attribute (diagnosis).

2. Linkage Attack Implementation and Results

Attack Methodology

The attack's goal was to re-identify records in our de-identified med_data_di table by linking them to the fully identified work_data table.

We implemented the attack logic in a LinkageAttackService. This service queries both tables to find unique QI combinations. A successful match occurs when a (gender, postal_code) pair has a COUNT = 1 in the work_data table (linking it to a known name) and also has a COUNT = 1 in the med_data_di table (linking it to a diagnosis).

Attack Results

Our linkage attack was successful and demonstrated the vulnerability. By executing the attack against the 1000-record dataset, we found 682 successful re-identifications.

These matches prove that simple de-identification is insufficient, as we were able to link 682 known individuals from the work_data table directly to their private medical diagnoses.

3. K-Anonymity Implementation

To defend against the linkage attack, we implemented k-anonymity using the QIs: age, gender, and postal_code. The value for k was made configurable in application.properties, and we used k=2 for our tests.

Algorithm and Techniques

We implemented the Mondrian algorithm. This algorithm was chosen for its efficiency in handling multiple QIs simultaneously. Our implementation, found in the AnonymityService, combined the following techniques:

1. **Top/Bottom Coding (Pre-processing):** To handle outliers in the age QI, our ProcessRecord class first applied top-and-bottom coding, capping all ages to the range of 18 to 90.
2. **Generalization (Core Logic):** The Mondrian algorithm recursively "cuts" the dataset. Each resulting partition (an "Equivalence Class") is then generalized. For example, an age partition becomes a range (e.g., "30-39"), and postalCode values (which were pre-processed to "12xxxxxxx") are generalized to "*" if the partition contains multiple codes.
3. **Suppression (Post-processing):** After the recursive partitioning, any residual group with fewer than k records was considered unsafe and was fully suppressed by setting all its QIs to “*”.

At the end of the k-anonymization process, we saved the k-anonymized dataset to a csv file.

Results and Privacy vs. Utility

Then, we executed our linkage attack against this new, protected table and got 0 successful re-identifications.

The attack found zero matches, confirming the k-anonymity implementation was successful in defending against our specific linkage attack. This success, however, comes at a significant cost to data utility. The generalization required to achieve k=2 (many postalCode and gender values became "*") renders the data far less useful for granular analysis.

4. Differential Privacy

This phase of the project allowed for the user to make requests to the med_data database and access information without exposing individual data. This was achieved with principles of Differential Privacy, by managing a privacy budget ϵ and adding Laplace noise to the true results. The goal was not just to return noisy answers, but to demonstrate the fundamental privacy-utility trade-off.

Queries Implemented:

- **Count** (GET /api/dp/count) of all users. This had the lowest possible sensitivity Δf=1.

- **Average of all users age** (GET /api/dp/average). This calculates NoisySum/Noisycount and has a sensitivity Δf = 122 due to the SUM(age) operation, which required clipping the maximum age by 0 and 122.
- **Histograms:**
    - AVG(age) group by age.
    - Age count divided into categories.
    - Gender count for each gender.

The Privacy-Utility Trade-off: An Experimental Analysis

The core of this implementation was discovering that not all "grouped queries are equal. The utility of the result depends on the number of unique categories of the attribute we group by.

First, we tried to implement the query AVG (age) grouped by diagnosis but the problem was that the diagnosis attribute has a very high cardinality (992 unique values). To run this query even once would cost a lot of privacy budget which was not doable so we deleted and decided to use attributes with less unique values.

This experiment successfully demonstrated that providing privacy for high-cardinality grouped queries destroys their utility, while low-cardinality queries can provide very accurate results while still being formally private.