Incognito: Efficient Full-Domain K-Anonymity

Introduction

Incognito: Efficient Full-Domain K-Anonymity is an innovative algorithm designed to ensure the privacy of sensitive data while maintaining its utility for analysis. The primary goal of this algorithm is to achieve effective k-anonymization of data, ensuring that each record in the dataset is indistinguishable from at least k-1 other records. Using a tree-based approach, Incognito generalizes the data in a way that preserves relationships between them while maintaining a high level of anonymity. This algorithm provides a reliable and scalable solution to safeguard data privacy without compromising its utility for analysis and research.

Pseudocode

Input: A table T to be k-anonymized, a set Q of n quasi-identifier attributes, and a set of dimension tables (one for each quasi-identifier in Q)
Output: The set of k-anonymous full-domain generalizations of T

```
C1 = {Nodes in the domain generalization hierarchies of attributes in Q}
E1 = {Edges in the domain generalization hierarchies of attributes in Q}
queue = an empty queue
for i = 1 to n do
  //Ci and Ei define a graph of generalizations
  Si = copy of Ci
  {roots} = {all nodes ∈ Ci with no edge ∈ Ei directed to them}
  Insert {roots} into queue, keeping queue sorted by height
  while queue is not empty do
    node = Remove first item from queue
    if node is not marked then
      if node is a root then
        frequencySet = Compute frequency set of T with respect to attributes of node using T.
      else
        frequencySet = Compute frequency set of T with respect to attributes of node using parent's frequency set.
      end if
      Use frequencySet to check k-anonymity with respect to attributes of node
      if T is k-anonymous with respect to attributes of node then
        Mark all direct generalizations of node
      else
        Delete node from Si
        Insert direct generalizations of node into queue, keeping queue ordered by height
      end if
    end if
  end while
  Ci+1, Ei+1 = GraphGeneration(Si, Ei)
end for
return Projection of attributes of Sn onto T and dimension tables
```

Implementation

Development Environment:

- **Programming Language**: Python
- **Database**: SQLite
- **Tools**: Python libraries such as `sqlite3`, `pandas`, `csv`, `argparse` for data manipulation and command-line argument management. The libraries are directly used in a standard development environment.

Choice of SQLite as Database:

- **Reasons for the Choice**: SQLite was selected for its lightweight nature, ease of use, and portability. Additionally, its direct integration with Python through the `sqlite3` library simplifies interaction with the database from code.
- **Volatility**: The database is designed to be volatile, meaning it is used primarily for temporary storage during the execution of the program. Data is not meant to be persisted long-term, which allows for quicker setups and teardowns and reduces overhead.

Use of Tables for Candidate Nodes and Edges:

- **Candidate Nodes Table (Table C)**: Stores information about candidate nodes C, such as ID, dimensions, generalization index and parents.
- **Edges Table (Table E)**: Contains information about edges between candidate nodes (id start, id end).

Design Choices Rationale:

- **Use of a Database for Queries**: SQLite provides query efficiency and structured data management.
- **Separation of Graph Data into Distinct Tables**: This allows for more efficient graph management and simplifies operations like the addition, modification, and deletion of nodes and edges.

Data Preparation:

- The script starts by reading the main dataset from a CSV file using the `prepare_table_for_k_anonymization` function. This function creates an SQLite table to store the dataset's contents. It also ensures that attribute names are properly formatted and inserts the data into the table.

Creation of Dimension Tables:

- The dimension tables are essential for managing the sensitivity of the data. Each dimension table represents a quasi-identifier attribute and its associated generalization hierarchy. The `get_dimension_tables` function reads CSV files containing dimension data and creates corresponding dimension tables in SQLite.

Anonymization Algorithm:

- The anonymization algorithm is based on the Incognito algorithm, a popular approach for achieving k-anonymity. It consists of several phases:
    - **Initialization**: Creation of the initial candidate nodes (`C1`) representing the root nodes of the generalization hierarchies.
    - **Basic Incognito Algorithm**: Iteratively processes each quasi-identifier attribute hierarchy to anonymize the data. This involves identifying candidate nodes, determining anonymity, and applying generalization as needed.
    - **Graph Generation**: Generates candidate nodes and edges for the next iteration of the algorithm, ensuring the completeness of the generalization process.
    - The algorithm maintains a priority queue to process nodes based on their height in the generalization hierarchy. It ensures that nodes are processed in the correct order to maintain data relationships.

Projection of Anonymized Data:

- Once anonymization is complete, the script projects the anonymized data onto a new CSV file. This projection involves selecting relevant attributes from the anonymized dataset and storing them in a new file for further analysis or storage.

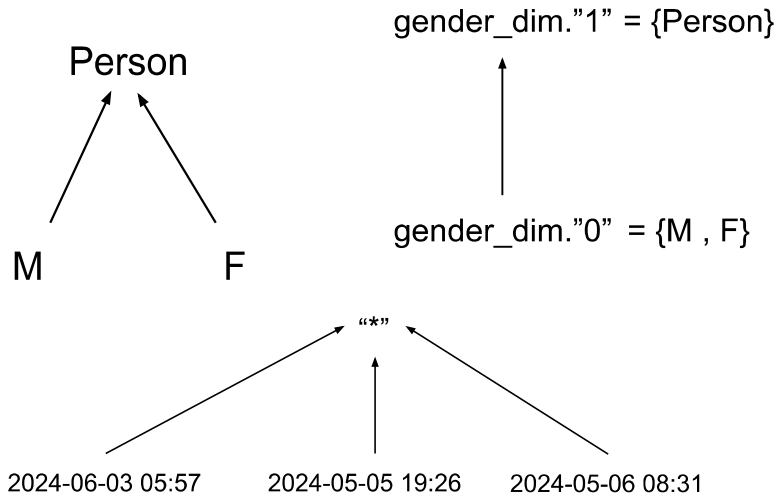
Customizable Parameters:

- The script allows users to specify parameters such as the k value for k-anonymity and a threshold t value for ignoring outliers. These parameters can be adjusted to balance privacy and data utility according to specific requirements.

Exception Handling:

- The script includes exception handling to deal with potential errors, such as invalid parameter values or issues with data processing. This ensures robustness and reliability in handling various scenarios during the anonymization process.

Hierarchical structure of dimensions tables provided as input



age_dim.”3” ={40, 60, 80} same value every 20 values

age_dim.”2” ={40, 60, 90} same value every 10 values

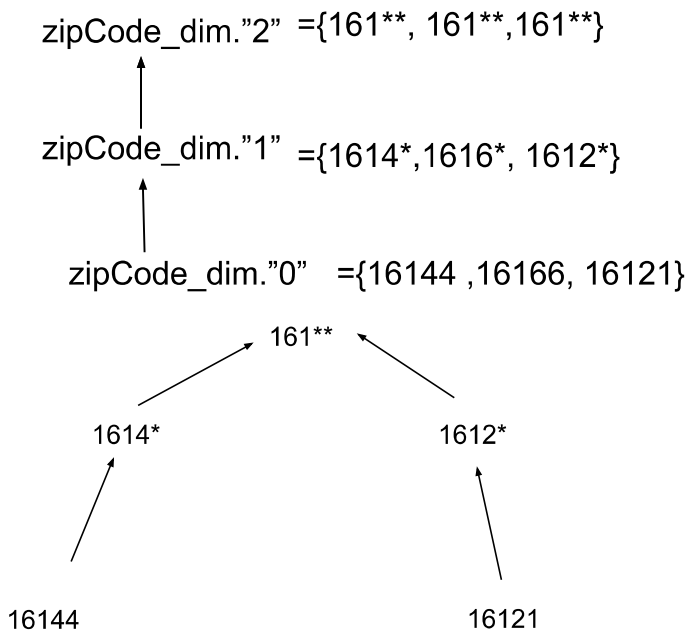
age_dim.”1” ={45, 65, 95} same value every 5 values

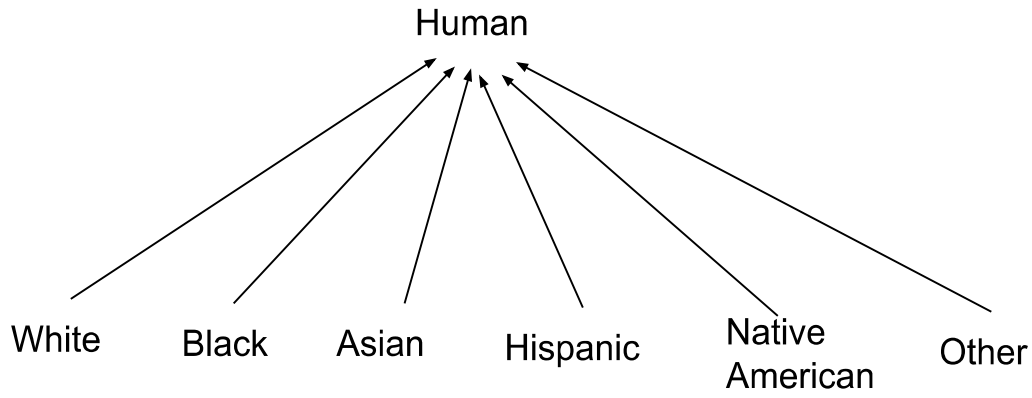
age_dim.”0” ={46, 69, 97}

height_dim.”2”={140, 160,180} set of 20 values

height_dim.”1”={150,170, 190} same value every 10 values

height_dim.”0” ={155,173, 199}





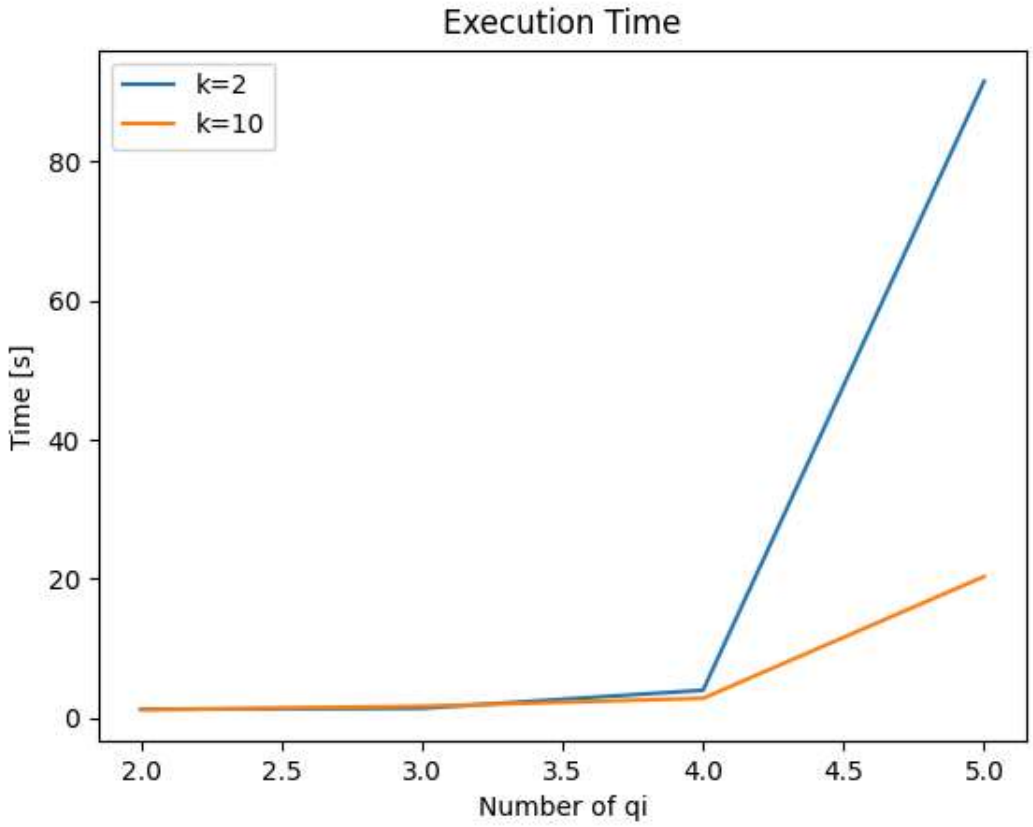
weight_dim.”2” ={40, 60,100} same value every 20 values

weight_dim.”1” ={50,70, 110} same value every 10 values

weight_dim.”0” ={51,78, 119}

Analysis of Execution Performance and Results

- efficiency and complexity



The execution time exhibits exponential growth with respect to the increase in **QI** values from 2 to 5 across 5 runs with a dataset containing 1000 tuples, indicating that the complexity of the algorithm is **exponential**. This corresponds to the expected complexity, which is exponential with respect to the number of quasi-identifiers.

I have observed that in my execution with k=10, the growth curve is less steep compared to k=2.

The algorithm is efficient for handling large amounts of data and large databases. However, this efficiency diminishes as the number of quasi-identifiers (QIs) increases.

Privacy-utility level

After performing an analysis on the anonymized dataset to the previous execution and following the metrics of Discernibility Metric (DM) and Normalized Certainty Penalty (NCP), I have observed the following values:

k = 2
../datasets/anonymous_table_k_2.csv
Discernibility Metric (DM): 1030
Normalized Certainty Penalty (NCP): 0.6137980753276919
k = 10
../datasets/anonymous_table_k_10.csv
Discernibility Metric (DM): 1052
Normalized Certainty Penalty (NCP): 0.6050872739339638

These metrics suggest that the anonymization process resulted in a moderate level of data indistinguishability and a significant, but not extreme, loss of information.

I only anonymized part of the data, which didn't improve privacy much. This made it easy to still tell different pieces of data apart, even when I tried to group them. However, the actual amount of information I lost didn't change much. This suggests that the parts of the data we didn't anonymize are keeping a lot of the original information. So to make the data more private, we need to anonymize more of it, so it's harder to distinguish individual pieces of information.

hospital.csv

patientId,dateandTime,age,height,gender,race,weight,zipCode,Hypertension,Arthritis,Asthma,Pneumonia,Diabetes

630239,2024-06-03 05:57,20,165,M,Black,40,16163,0,0,1,0,0

628350,2024-05-05 19:26,20,183,M,White,55,16155,0,0,0,0,0

643771,2024-05-06 08:31,80,170,M,White,48,16150,0,0,0,0,1

633088,2024-04-29 12:35,92,164,F,White,62,16128,0,1,1,1,0

662886,2024-04-18 01:34,59,157,F,White,43,16158,0,0,1,1,0

612210,2024-04-30 14:58 ,53,178,M,White,40,16129,1,0,0,0,0

661816,2024-05-01 20:43 ,21,166,F,White,42,16153,0,1,0,0,1

688201,2024-05-22 23:15 ,59,178,M,White,57,16130,0,0,1,0,0

692275,2024-05-04 18:26 ,89,184,F,White,68,16130,0,1,0,1,0

anonymous_table_k_2.csv:

patientId,dateandTime_dim.'1',age_dim.'1',height_dim.'2',gender_dim.'1',race_dim.'1',weight, zipCode, …SD …

630239,*,20,160,Person,Human,40,16163,0,0,1,0,0

628350,*,20,180,Person,Human,55,16155,0,0,0,0,0

643771,*,80,160,Person,Human,48,16150,0,0,0,0,1

633088,*,90,160,Person,Human,62,16128,0,1,1,1,0

662886,*,55,140,Person,Human,43,16158,0,0,1,1,0

612210,*,50,160,Person,Human,40,16129,1,0,0,0,0

661816,*,20,160,Person,Human,42,16153,0,1,0,0,1

688201,*,55,160,Person,Human,57,16130,0,0,1,0,0

692275,*,85,180,Person,Human,68,16130,0,1,0,1,0

anonymous_table_k_10.csv:

patientId,dateandTime_dim.'1',age_dim.'2',height_dim.'2',gender_dim.'1',race_dim.'1',weight, zipCode, …SD…

630239,*,20,160,Person,Human,40,16163,0,0,1,0,0

628350,*,20,180,Person,Human,55,16155,0,0,0,0,0

643771,*,80,160,Person,Human,48,16150,0,0,0,0,1

633088,*,90,160,Person,Human,62,16128,0,1,1,1,0

662886,*,50,140,Person,Human,43,16158,0,0,1,1,0

612210,*,50,160,Person,Human,40,16129,1,0,0,0,0

661816,*,20,160,Person,Human,42,16153,0,1,0,0,1

688201,*,50,160,Person,Human,57,16130,0,0,1,0,0

692275,*,80,180,Person,Human,68,16130,0,1,0,1,0

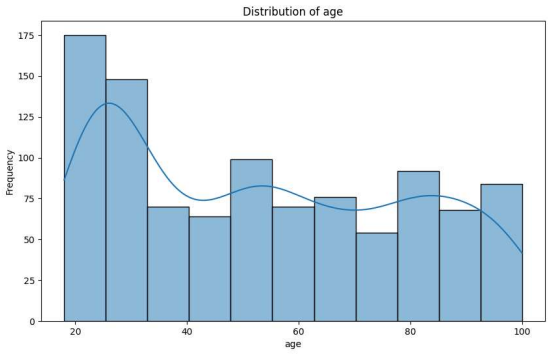
Statistical information before/after

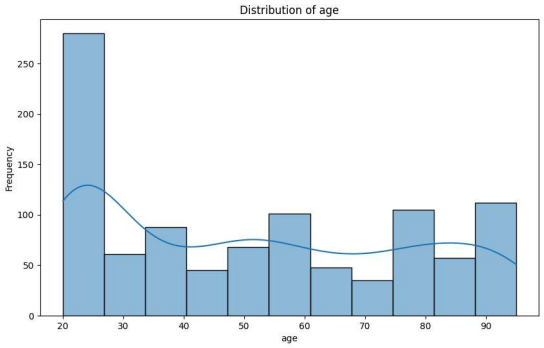
Considering the execution taken into analysis previously

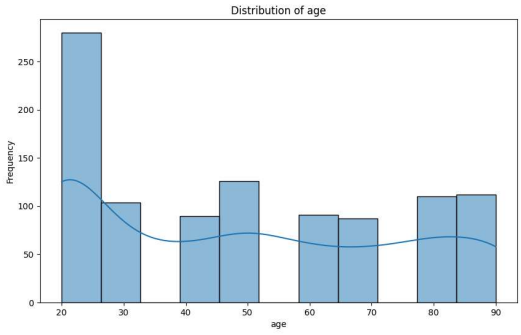
the anonymization slightly altered the data statistics. For example:

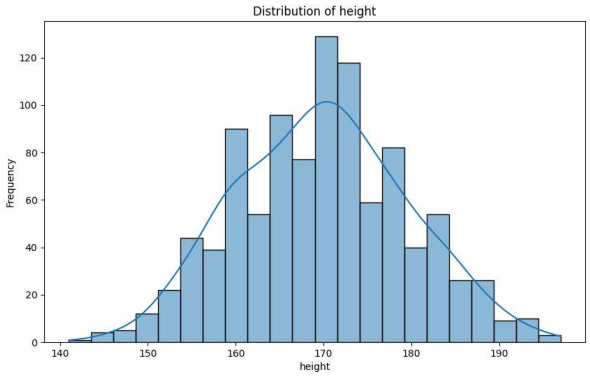
- In the original dataset:
  - Mean age: 53.379
  - Mean height: 169.807
- After anonymization with k=2
  - Mean age: 51.525
  - Mean height: 160.100
- After anonymization with k=10
  - Mean age: 49.050
  - Mean height: 160.100

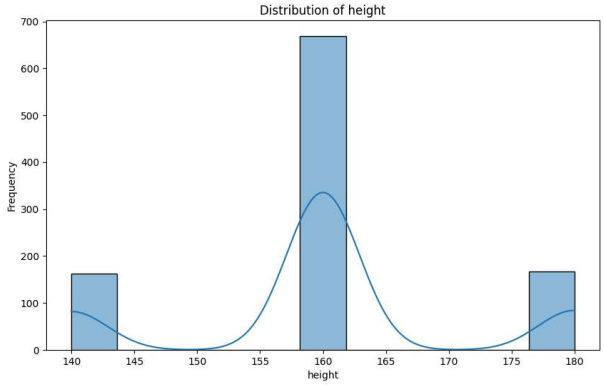
Additionally, the anonymization led to changes in standard deviation, quartiles, and the loss of detailed gender and race information, which became generic.











Conclusion

Incognito is a complete solution for producing k-anonymous full-domain generalizations, using the two key ideas of bottom-up aggregation along generalization dimensions and a priori computation. However our algorithm is ultimately exponential in the size of the quasi identifier.
For future data work, we should check different ways to keep information private than using Incognito.
Incognito is good for privacy, but it is slow. We need to optimize it like using "super-roots Incognito" or “cube-incognito“ as mentioned in the paper, or try other ways to keep data privacy while still doing good analysis.