DATASHIELD – ENSURING PRIVACY WITH K-ANONYMITY PROJECT REPORT

OF

DATABASE AND ONLINE SOCIAL MEDIA SECURITY
(CSLM 654)

MASTER OF TECHNOLOGY

In

COMPUTER SCIENCE & ENGINEERING

Submitted By

ARWAZ KHAN & ARYAM SHRIVASTVA (242210005 & 242210006)

Submitted To

DR. SHELLY SACHDEVA (ASSOCIATE PROFESSOR, DoCSE)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING NATIONAL INSTITUTE OF TECHNOLOGY DELHI MAY 2025

TABLE OF CONTENTS

S. No.	Title	Page No.
1	What is k-Anonymity	3
2	When is k-Anonymity Used	3
3	How does k-Anonymity Work	3
4	Example of k-Anonymity Work	4
5	Limitations of k-Anonymity	4
6	Core Components & Working	5
7	Advantages & Disadvantages	5
8	Comparison & Application of k-Anonymity	6
9	Software and Hardware Requirements	7
10	DataShield – K-Anonymity Application Algorithm	7
11	Code	9
12	Screenshots	18
13	Future Work	21
14	Reference	22

1. What is k-Anonymity?

k-Anonymity is a privacy-preserving technique used in data publishing to prevent the re-identification of individuals in datasets. It ensures that any individual cannot be distinguished from at least k − 1 other individuals based on a set of quasi-identifiers (QIDs). A dataset satisfies k-anonymity if every combination of quasi-identifier attributes occurs in at least k records.

- Quasi-identifiers: Attributes like age, ZIP code, or gender that may not uniquely identify someone on their own but can do so when combined.
- Anonymized records: By generalizing or suppressing QIDs, the dataset ensures that each person's record is indistinguishable from at least k−1 others.

2. When is k-Anonymity Used?

- When releasing datasets for research or statistical purposes while preserving user privacy.
- In healthcare, finance, or government records, where sensitive data must be protected from re-identification.
- To comply with privacy regulations like GDPR or HIPAA.
- When publishing public datasets for data mining, machine learning, or academic use.

3. How does k-Anonymity Work?

1. **Identify Quasi-Identifiers (QIDs):**
   - Detect which attributes could be used to identify individuals when combined with external information.
2. **Generalization and Suppression:**
   - Generalize specific values (e.g., age 28 → 20–30).
   - Suppress values where generalization is insufficient.
3. **Group Records:**
   - Modify the dataset such that for every set of QIDs, there are at least **k** identical records.
4. **Check Anonymity:**
   - Ensure that every record is indistinguishable from at least **k − 1** others based on QIDs.

4. Example of k-Anonymity Work

Age	ZIP Code	Disease
25	13053	Flu
27	13068	Cold
29	13053	Cancer

After 3-Anonymity:

Age	ZIP Code	Disease
25	13***	Flu
27	13***	Cold
29	13***	Cancer

Now, any individual cannot be re-identified since each row shares QID values with at least two others (k=3).

5. Limitations of k-Anonymity

- **Homogeneity Attack:** All records in a group have the same sensitive value, making inference easy.
- **Background Knowledge Attack:** If an attacker knows additional information, k-anonymity may still leak data.
- Does not protect against attribute disclosure, only identity disclosure.

To address these, more advanced techniques like l-diversity and t-closeness have been introduced.

6. Core Components & Working

Core Components of k-Anonymity:

1. **Quasi-Identifiers (QIDs):** These are indirect identifiers such as age, gender, and ZIP code that, when combined, can uniquely identify individuals. They are the main focus of anonymization in datasets.
2. **Generalization Module:** This module transforms specific data values into broader ranges or categories. For example, an age of 27 may be generalized to a range like 20–30. This helps group similar records together.
3. **Suppression Module:** When generalization isn't sufficient, sensitive or identifying data values are suppressed (replaced with a * symbol or removed entirely) to meet privacy requirements.
4. **Anonymization Engine:** It processes the dataset using the generalization and suppression techniques to ensure that for every set of QID values, there are at least k identical records.
5. **Evaluation Mechanism:** After anonymization, this component verifies whether the dataset satisfies k-anonymity, i.e., each record is indistinguishable from at least k−1 others based on the QIDs.

Working of k-Anonymity Process:

- The system begins by identifying the QIDs in a dataset.
- Then, it applies generalization and suppression to these fields to minimize identifiability.
- Records are grouped so that each group shares the same QID values with at least k records.
- Finally, the processed dataset is evaluated to ensure compliance with k-anonymity, and is then exported.

7. Advantages & Disadvantages

Advantages:

- **Simplicity:** Easy to understand and apply using standard data transformation methods.
- **Privacy Assurance:** Ensures that each person in the dataset cannot be uniquely identified.

- **Regulatory Compliance:** Meets basic privacy requirements under laws like GDPR, HIPAA.
- **Wide Applicability:** Useful across sectors like healthcare, finance, and public policy.

Disadvantages:

- **Homogeneity Attack Risk:** If all records in a group have the same sensitive value, an attacker can still infer information.
- **Background Knowledge Attack:** Attackers with external knowledge may still identify individuals.
- **Information Loss:** Generalization and suppression may reduce the dataset's utility.
- **Scalability Issues:** Difficult to maintain data utility as the dataset grows or becomes high-dimensional.

8. Comparison & Application of k-Anonymity

Comparison with l-Diversity and t-Closeness:

- **k-Anonymity:**
  Protects against identity disclosure by ensuring each record is similar to at least k-1 others in terms of QIDs.
- **l-Diversity:**
  Extends k-anonymity by requiring diversity in sensitive attribute values within each group. This prevents inference of the sensitive value even if a group is identified.
- **t-Closeness:**
  Ensures the distribution of sensitive attributes in each group closely matches the distribution in the entire dataset, offering even stronger protection.

Applications of k-Anonymity:

- **Public Health Data Sharing:** Used to anonymize medical records before sharing them for research.
- **Census Data Publication:** Ensures population data is made available without risking individual identities.
- **Data Mining and Machine Learning:** Allows training models on anonymized datasets while maintaining user privacy.
- **Social Media and Behavioral Analytics:** Enables safe publication of user interaction data for academic or commercial use.

9. Software and Hardware Requirements

Category	Requirement
Software	
Operating System	Windows 10/11 or Linux-based OS
Python Version	Python 3.8 or above
Web Framework	Flask
Templating Engine	Jinja2
Database	SQLite (temporary storage)
Text Editor/IDE	Visual Studio Code (VS Code)
Browser	Chrome / Firefox (for running the web app)
Hardware	
Processor	Intel i3 or above
RAM	Minimum 4 GB
Storage	Minimum 500 MB free space
Display	13” or larger recommended (for development view)

10. DataShield – K-Anonymity Application Algorithm

1. Initialization Phase

1.1 App Initialization (app.py)

- Initialize Flask app with a secret key.
- Import required modules:
  - Flask, render_template, request, jsonify, send_file
  - pandas, io for in-memory file handling
- Define apply_k_anonymity(df, k, columns):
  - For each specified column:
    - Mask last k characters with *
    - If length < k, mask the whole value

1.2 Frontend Setup

- index.html provides:
  - Form inputs: CSV file, k-value, column names
  - Linked to script.js for handling submission
  - Uses style.css for UI styling

2. User Interaction Flow

2.1 Initial Load

- **Route /**:
    - Renders index.html

2.2 Upload & Processing

- **Form Action /upload**
    - Method: POST
    - Triggered via JavaScript handleFormSubmit(event)
    - Executes SweetAlert2 loading animation

3. Backend Processing Algorithm

3.1 Route: /upload (POST)

1. Check for file presence in request
2. If file is missing or filename is empty:
    - Return JSON error response
3. Retrieve k_value and columns from form
4. Load CSV file into pandas DataFrame
5. Apply apply_k_anonymity function
6. Convert processed DataFrame to CSV in-memory
7. Return anonymized CSV file using send_file

4. Frontend JavaScript Handling (script.js)

handleFormSubmit(event)

1. Prevent default form submission
2. Show loading alert via SweetAlert2
3. Prepare FormData from the form
4. Send POST request to /upload with form data
5. On success:
    - Convert response to blob
    - Trigger download of anonymized_file.csv
    - Show success message
6. On error:
    - Show error alert with reason

5. User Interface Design (style.css)

- Background: full-screen image with centered container
- Form:
  - Styled file input, number input, text input
  - Submit button styled with hover effect
- Container:
  - Semi-transparent background
  - Responsive and centered

6. Summary of File Flow

File	Purpose
app.py	Flask backend for processing and routing
index.html	Main interface for CSV upload
script.js	Handles form submission and download logic
style.css	Custom UI design and layout

11. Code

Code: app.py

```
from flask import Flask, render_template, request, jsonify, send_file
import pandas as pd
import io

app = Flask(__name__)

# Simple k-anonymity function
def apply_k_anonymity(df, k, columns):
    for col in columns:
        df[col] = df[col].astype(str).apply(lambda x: x[:len(x)-k] + '*'*k if len(x) > k else
'*'*len(x))
    return df

@app.route('/')
def index():
    return render_template('index.html')
```

```
@app.route('/upload', methods=['POST'])
def upload_file():
    if 'file' not in request.files:
        return jsonify({"error": "No file part"}), 400

    file = request.files['file']
    if file.filename == '':
        return jsonify({"error": "No selected file"}), 400

    try:
        # Get form values
        k = int(request.form['k_value'])
        columns = request.form['columns'].replace(" ", "").split(",")

        # Read the uploaded file into a DataFrame
        df = pd.read_csv(file)
        df_anonymized = apply_k_anonymity(df, k, columns)

        # Convert DataFrame to CSV (in memory, no saving)
        output = io.StringIO()
        df_anonymized.to_csv(output, index=False)
        output.seek(0)

        return send_file(
            io.BytesIO(output.getvalue().encode()),
            as_attachment=True,
            mimetype="text/csv",
            download_name=f"anonymized_{file.filename}"
        )

    except Exception as e:
        return jsonify({"error": str(e)}), 500


if __name__ == '__main__':
    app.run(debug=True)
```

Explanation:

1. **apply_k_anonymity(df, k, columns)**
   Implements a basic k-anonymity transformation by masking the last k characters in each specified column using asterisks. Operates directly on a Pandas DataFrame and supports string conversion for mixed-type data.

2. **index()**
   Root route controller rendering the HTML form (index.html) for uploading CSV files and entering the k-value and columns to be anonymized. It initializes user interaction with the app.

3. **upload_file()**
   Upload and processing handler responsible for:
   - Validating presence of the uploaded file and form fields.
   - Parsing the k-value and column list.
   - Reading the file as a CSV DataFrame.
   - Applying apply_k_anonymity() to anonymize specified columns.
   - Returning the processed file as a downloadable CSV via memory buffer. Handles all processing in-memory without file system dependency.

4. **__main__ block (if __name__ == '__main__':)**
   Starts the Flask development server with debug=True, allowing auto-reload and detailed error tracing during development.

Code: index.html

```
<!DOCTYPE html>
<html>

<head>
   <title>DataShield – Ensuring Privacy with K-Anonymity</title>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <link rel="icon" href="{{ url_for('static', filename='favicon.png') }}"
type="image/icon">
   <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">

   <!-- SweetAlert2 CDN -->
   <script src="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>
</head>
```

```
<body>
  <div class="container">
    <h3>Upload CSV for K-Anonymity Processing</h3>
    <form action="/upload" method="post" enctype="multipart/form-data"
onsubmit="handleFormSubmit(event)">
      <label for="file">Choose CSV File:</label>
      <input type="file" name="file" required>

      <label for="k_value">Enter k-value:</label>
      <input type="number" name="k_value" min="1" required>

      <label for="columns">Enter Columns (comma-separated):</label>
      <input type="text" name="columns" placeholder="e.g., Name, Age" required>

      <input type="submit" value="Upload and Process">
    </form>
  </div>

  <script src="{{ url_for('static', filename='script.js') }}"></script>
</body>

</html>
```

Explanation:

1. **Document Declaration**
   Declares an HTML5 document using <!DOCTYPE html>, preparing the browser to parse content as modern HTML.

2. **Head Section**
   - Sets the title as "DataShield – Ensuring Privacy with K-Anonymity"
   - Defines UTF-8 encoding for proper character rendering
   - Enables mobile responsiveness via viewport meta tag
   - Loads favicon using Flask’s url_for('static', filename='favicon.png')
   - Links custom CSS (style.css) for styling the UI
   - Loads SweetAlert2 via CDN to enable modern popup alerts during form submission

3. **Body Section**
    - Contains a centered div.container styled via CSS for layout and theming
    - Displays a heading <h3> prompting users to upload a CSV for anonymization

4. **Form Definition (<form>)**
    - Action: Submits to /upload route
    - Method: POST, allowing data/file submission
    - enctype: multipart/form-data enables file upload
    - onsubmit: Triggers handleFormSubmit(event) from script.js to handle the upload asynchronously with alerts

5. **Form Fields**
    - File Input:
      Prompts user to upload a .csv file
      <input type="file" name="file" required>

    - K-Value Input:
      Accepts anonymization level k as a positive integer
      <input type="number" name="k_value" min="1" required>

    - Columns Input:
      Accepts comma-separated column names to be anonymized
      <input type="text" name="columns" required>

    - Submit Button:
      Submits form to start processing
      <input type="submit" value="Upload and Process">

    - JavaScript Inclusion
      Loads script.js using Flask’s url_for() to handle form submission logic, file download, and user notifications

Code: style.css

```
body {
   font-family: Arial, sans-serif;
   background-image: url('./bg.png');
   /* Add your image file */
   background-size: cover;
   background-position: center;
   background-repeat: no-repeat;
   text-align: center;
   margin: 0;
   padding: 0;
   height: 100vh;
   display: flex;
   justify-content: center;
   align-items: center;
}

.container {
   background: rgba(77, 108, 201, 0.5);
   /* Semi-transparent background */
   padding: 50px;
   border-radius: 10px;
   box-shadow: 0px 0px 10px rgba(35, 9, 206, 0.1);
   max-width: 400px;
   width: 100%;
}

h3 {
   color: #FFF;
}

label {
   font-weight: bold;
   display: block;
   margin-top: 10px;
   color: #FFF;
}

input[type="file"],
input[type="number"],
input[type="text"] {
   width: 100%;
   padding: 8px;
   margin-top: 5px;
   border: 1px solid #ccc;
   border-radius: 5px;
}

input[type="submit"] {
   background-color: #28a745;
   color: white;
   padding: 10px;
   border: none;
   border-radius: 5px;
   cursor: pointer;
   width: 100%;
   margin-top: 15px;
}

input[type="submit"]:hover {
   background-color: #218838;
}
```

Explanation:

1. **Body Styling:**
    - Centers content with flexbox and sets a full-screen background image.

2. **Container:**
    - Semi-transparent background, rounded corners, padding, and a shadow. Max width is 400px.

3. **Heading (h3):**
    - White text color for headings.

4. **Labels:**
    - Bold text with white color and top margin.

5. **Input Fields:**
   - Full width, padding, light gray border, and rounded corners.

6. **Submit Button:**
   - Green background, white text, full width, rounded corners, and pointer cursor on hover.
   - Darker green when hovered.

The design focuses on a clean, centered form with a modern, responsive layout.

Code: script.js

```
function handleFormSubmit(event) {
    event.preventDefault();

    Swal.fire({
        title: "Processing...",
        text: "Please wait while we process your file.",
        allowOutsideClick: false,
        didOpen: () => {
            Swal.showLoading();
        }
    });

    let form = event.target;
    let formData = new FormData(form);

    fetch(form.action, {
        method: "POST",
        body: formData
    })
        .then(response => {
            if (!response.ok) {
                throw new Error("File processing failed.");
            }
            return response.blob(); // Convert response to file blob
        })
        .then(blob => {
            let a = document.createElement("a");
            a.href = URL.createObjectURL(blob);
```

```
      a.download = "anonymized_file.csv"; // Set filename
      document.body.appendChild(a);
      a.click();
      a.remove();

      Swal.fire({
        title: "Success!",
        text: "Processing complete! Your file has been downloaded.",
        icon: "success"
      });

      form.reset();
    })
    .catch(error => {
      Swal.fire({
        title: "Error!",
        text: error.message,
        icon: "error"
      });
    });
}
```

Explanation:

1. **Prevent Default Form Submission:**
   - event.preventDefault() stops the form from submitting normally, allowing for custom handling.
2. **SweetAlert Loading Popup:**
   - Shows a loading modal (Swal.fire) with a message indicating the file is being processed.
   - allowOutsideClick: false prevents closing the alert by clicking outside.
   - didOpen: () => { Swal.showLoading(); } triggers the loading animation when the modal opens.
3. **Form Data Creation:**
   - Creates a FormData object from the form to collect the form data (including files).
4. **Fetch API to Submit the Form:**
   - fetch(form.action, { method: "POST", body: formData }) sends the form data to the server via a POST request.

5. **Handling the Response:**
    - If the server response is successful (response.ok), the response is converted into a file blob.
    - The file blob is then used to create a download link (<a> tag), which triggers the download of the file named "anonymized_file.csv".
6. **Success or Error Handling:**
    - If the file is processed successfully, a success message is displayed using SweetAlert.
    - If an error occurs, an error message is shown.
7. **Reset Form:**
    - After the file is successfully downloaded, the form is reset.

Key Features:

- Uses Swal.fire() to show modals (loading, success, error).
- File download triggered after successful processing.
- Error handling and feedback provided to the user throughout the process.

12. Screenshots

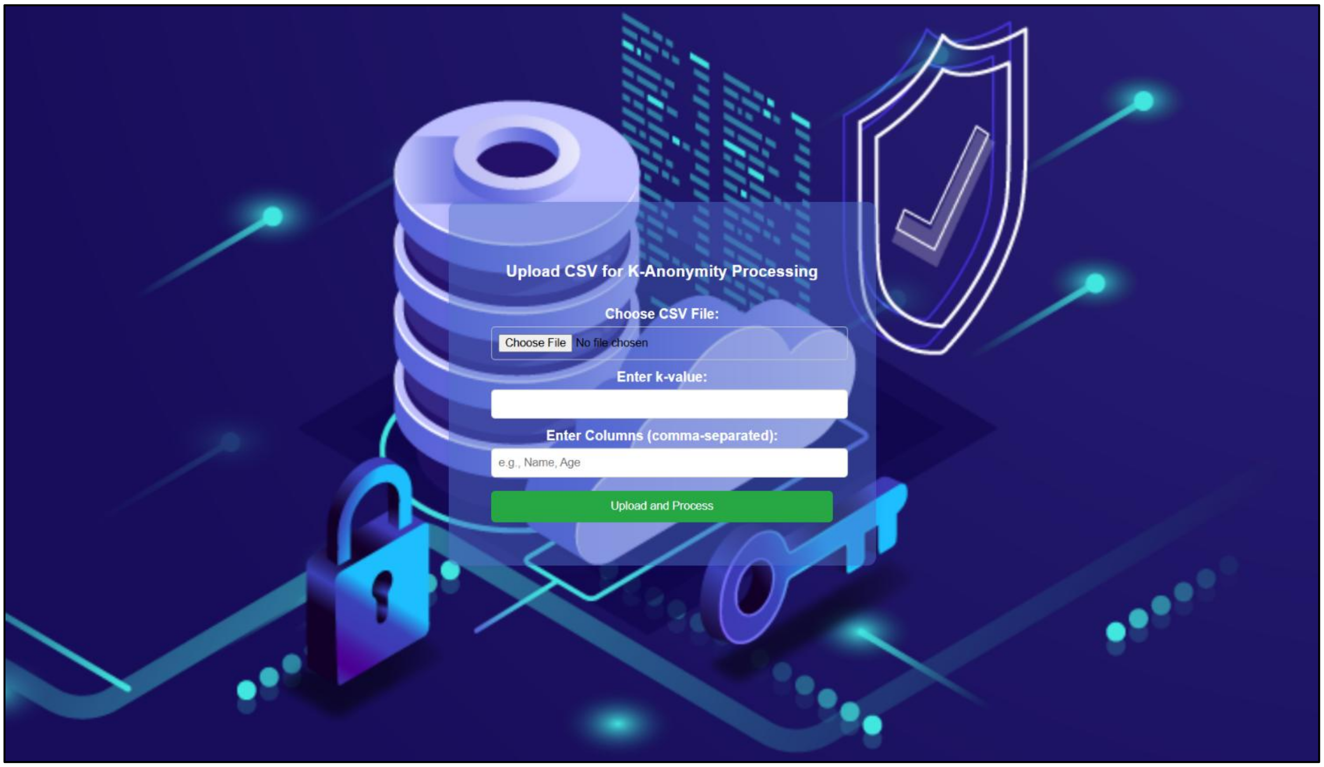
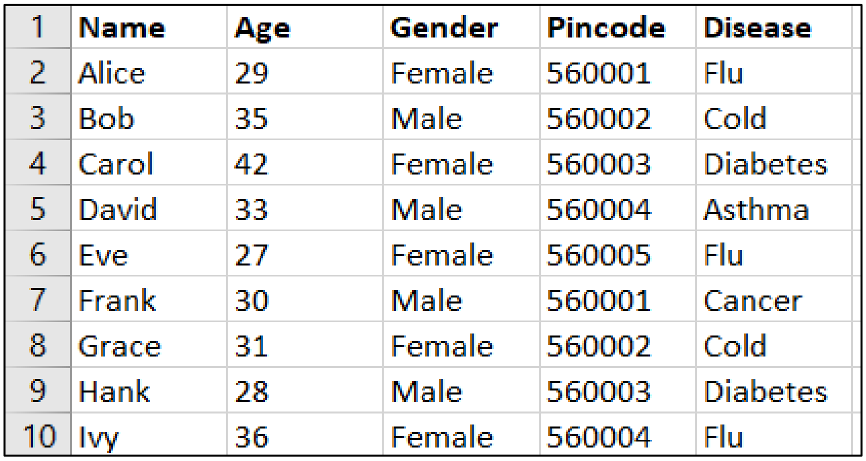
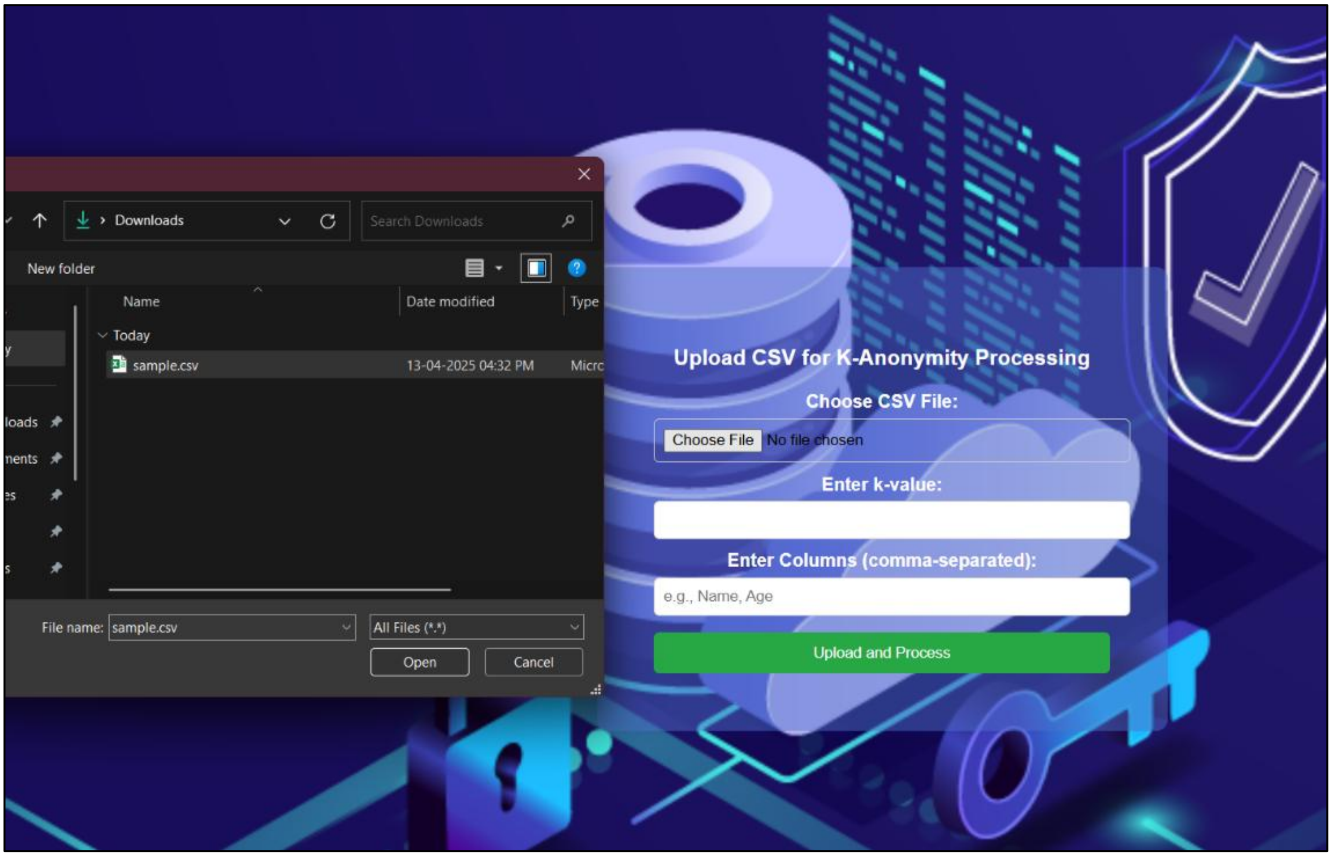


Fig 12.1: Web Application Interface



1	Name	Age	Gender	Pincode	Disease
2	Alice	29	Female	560001	Flu
3	Bob	35	Male	560002	Cold
4	Carol	42	Female	560003	Diabetes
5	David	33	Male	560004	Asthma
6	Eve	27	Female	560005	Flu
7	Frank	30	Male	560001	Cancer
8	Grace	31	Female	560002	Cold
9	Hank	28	Male	560003	Diabetes
10	Ivy	36	Female	560004	Flu

Fig 12.2: Sample Dataset Used for K-Anonymity



sample.csv 13-04-2025 04:32 PM Micro

File name: sample.csv

All Files (*.*)

Upload CSV for K-Anonymity Processing

Choose CSV File:

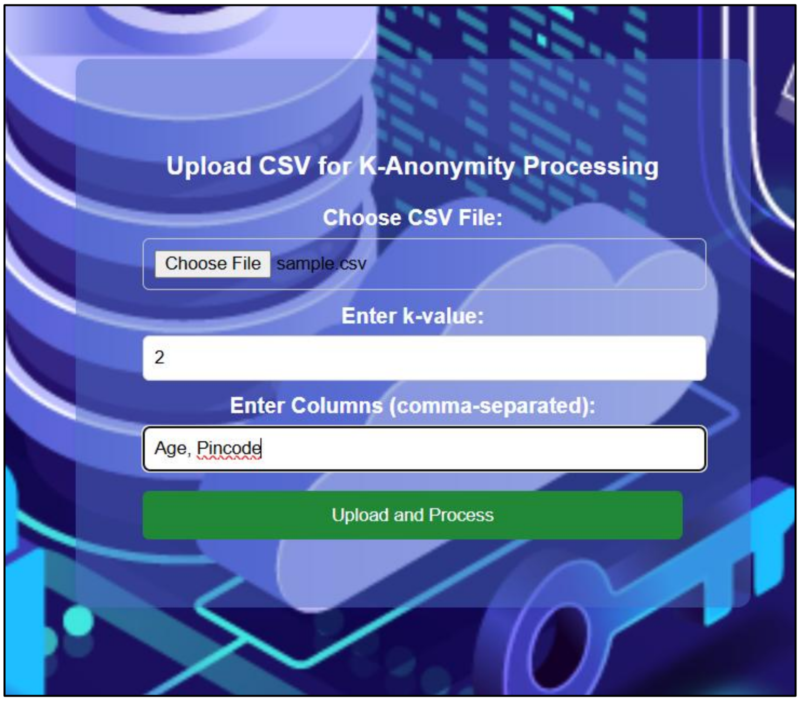
Choose File No file chosen

Enter k-value:

Enter Columns (comma-separated):

e.g., Name, Age

Fig 12.3: Selecting the Sample Dataset for Processing


Upload CSV for K-Anonymity Processing

Choose CSV File:

Choose File sample.csv

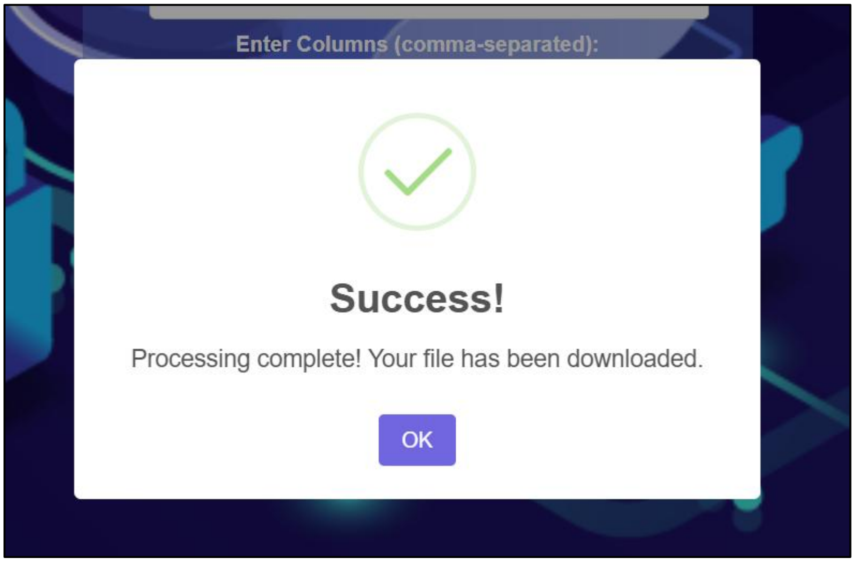
Enter k-value:

2

Enter Columns (comma-separated):

Age, Pincode

Fig 12.4: Defining the K-Value and Specifying Column Names for Anonymization


Enter Columns (comma-separated):

Success!

Processing complete! Your file has been downloaded.

Fig 12.5: Processed Dataset Downloaded Successfully

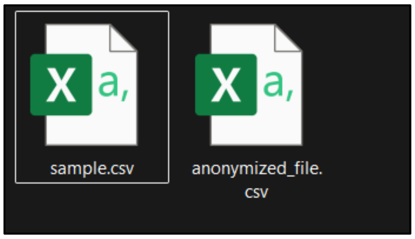
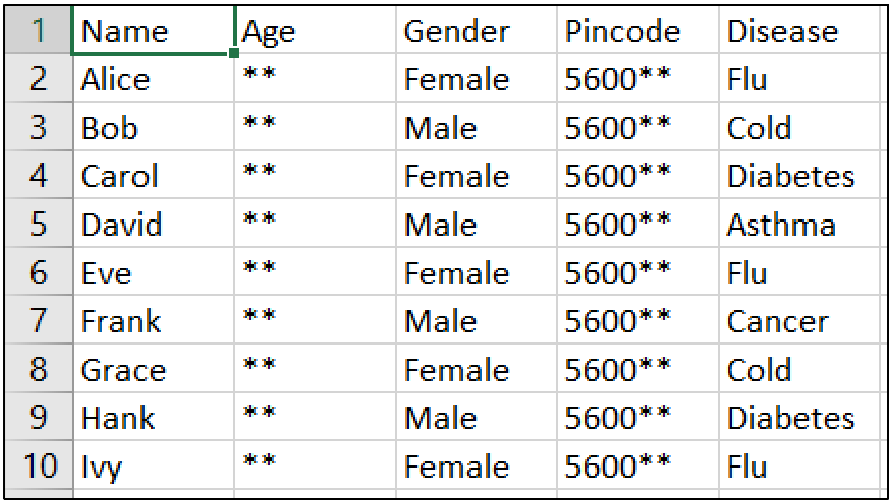


Fig 12.6: Downloaded Anonymized Dataset



1	Name	Age	Gender	Pincode	Disease
2	Alice	**	Female	5600**	Flu
3	Bob	**	Male	5600**	Cold
4	Carol	**	Female	5600**	Diabetes
5	David	**	Male	5600**	Asthma
6	Eve	**	Female	5600**	Flu
7	Frank	**	Male	5600**	Cancer
8	Grace	**	Female	5600**	Cold
9	Hank	**	Male	5600**	Diabetes
10	Ivy	**	Female	5600**	Flu

Fig 12.7: View of the Anonymized Dataset in CSV Format

13. Future Work

- Integrate advanced models like l-diversity and t-closeness to improve privacy.
- Enable real-time anonymization for streaming data.
- Improve scalability to handle large datasets efficiently.
- Allow user-defined privacy levels for flexible control.
- Incorporate privacy-preserving machine learning techniques.
- Enhance user interface and data visualization tools.
- Add evaluation metrics to balance privacy and utility.
- Ensure compliance with privacy laws like GDPR (General Data Protection Regulation).

14.References

[1] L. Sweeney, “k-Anonymity: A model for protecting privacy,” International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, vol. 10, no. 5, pp. 557–570, 2002.

[2] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam, “l-Diversity: Privacy beyond k-anonymity,” ACM Transactions on Knowledge Discovery from Data (TKDD), vol. 1, no. 1, pp. 3–es, 2007.

[3] R. Elmasri and S. B. Navathe, Fundamentals of Database Systems, 7th ed. Pearson, 2015.