# Leveraging Machine Learning for Enhanced Population Health Monitoring in LMICs: Linking Demographic Surveillance and Clinic Records

LONDON SCHOOL of HYGIENE &TROPICAL MEDICINE

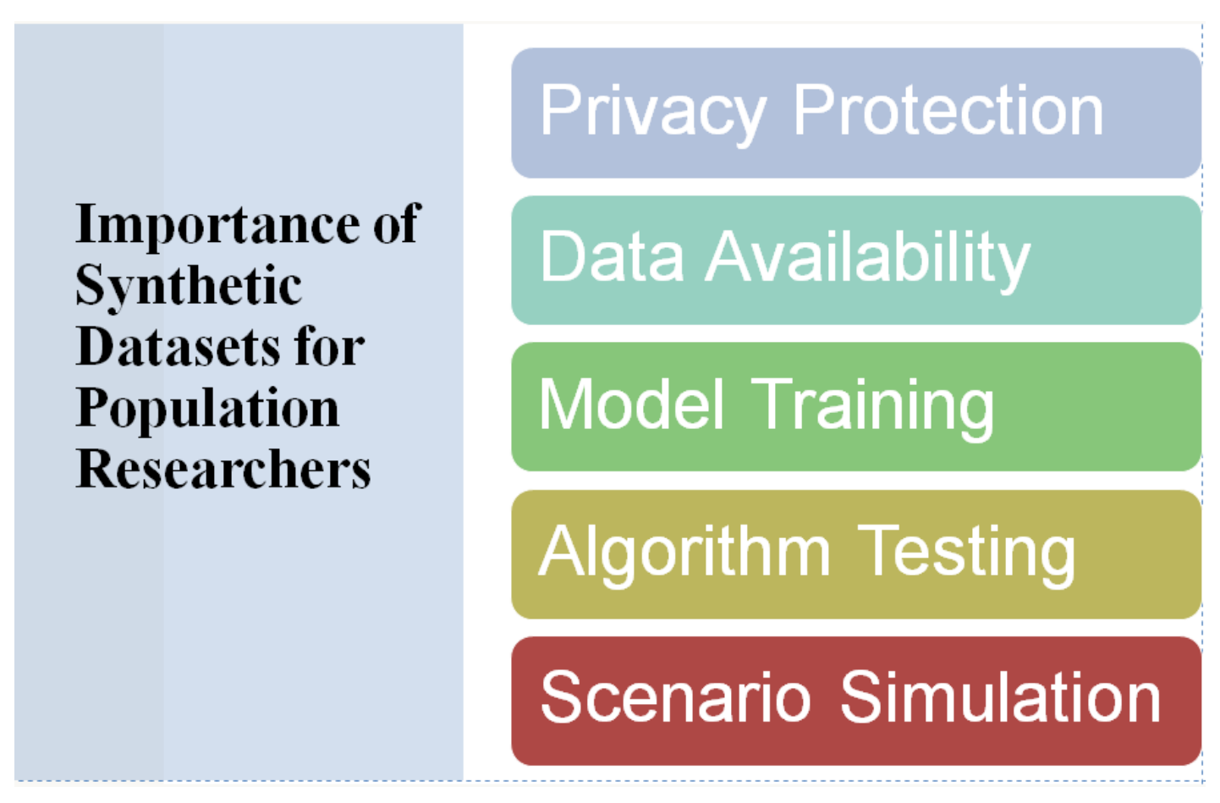
Tathagata Bhattacharjee[1], Emma Slaymaker[1], Chodziwadziwa Kabudula[2], Jim Todd[1]

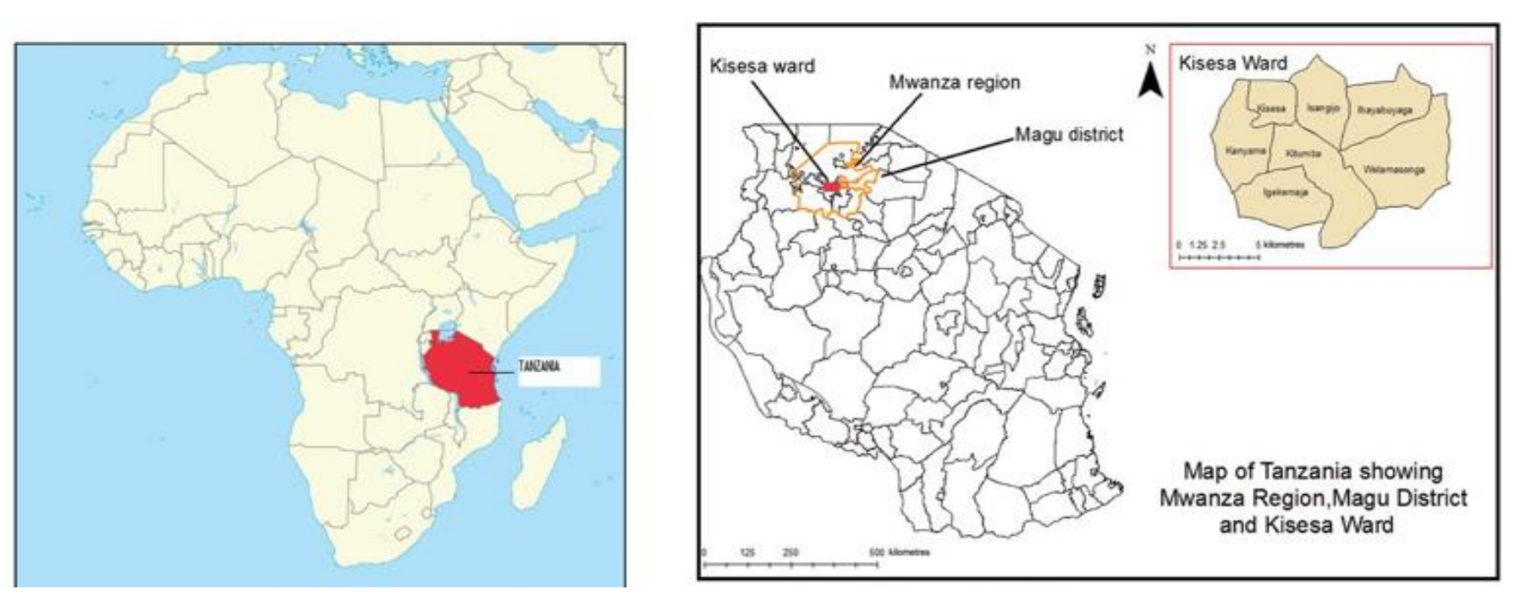
[1] *Department of Population Health, London School of Hygiene and Tropical Medicine, University of London, London, United Kingdom*
[2] *Medical Research Council/Wits Rural Public Health and Health Transitions Research Unit (Agincourt), South Africa*

Max Planck Institute for Demographic Research workshop: Demystifying machine learning for population researchers
November 5th - 6th, 2024, Rostock, Germany

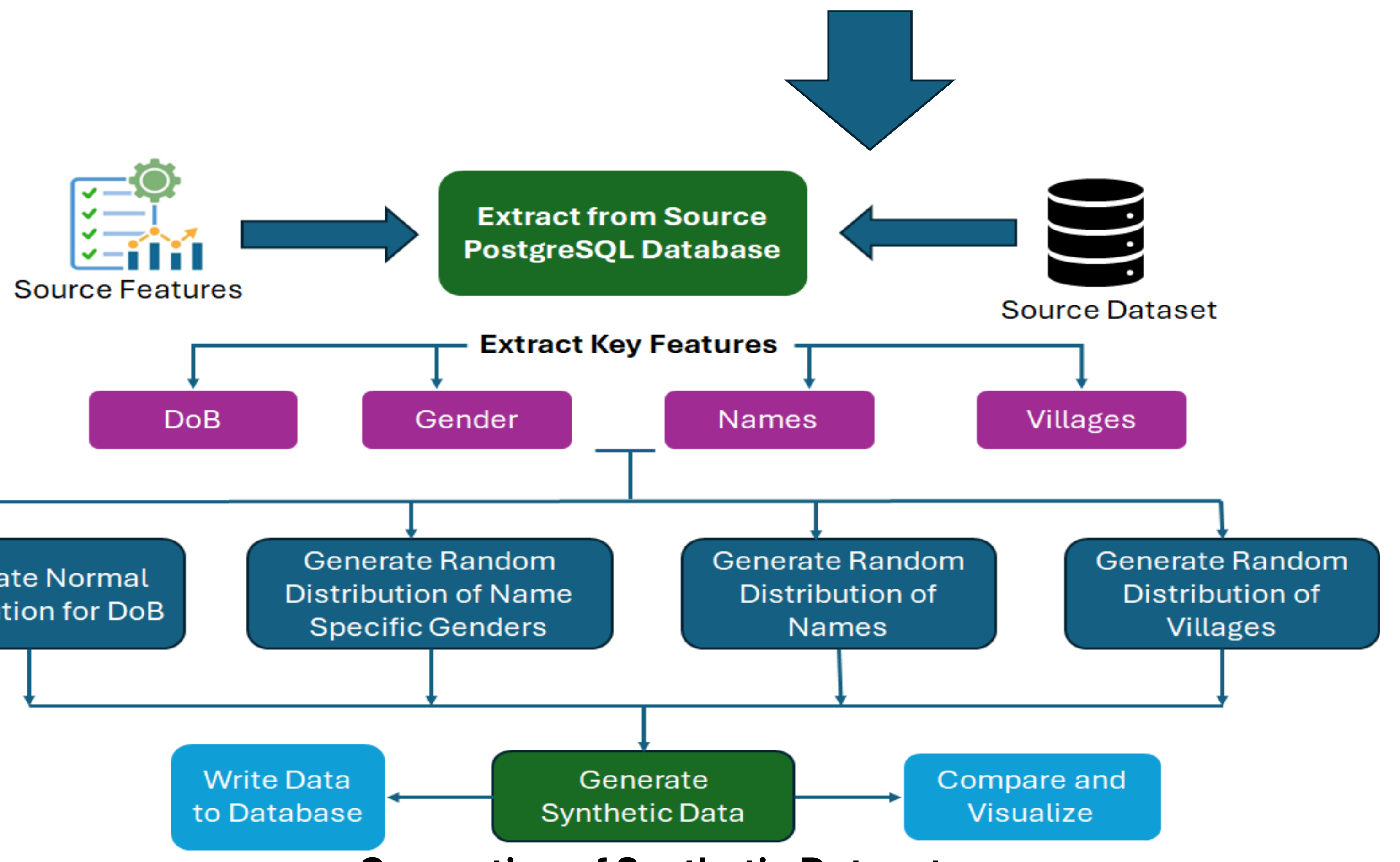
## Synthetic Data

Synthetic data mimics real-world data while protecting privacy, enabling researchers to train models and test algorithms without exposing sensitive information. It supports secure data sharing and collaboration, making it essential for advancing population health research.

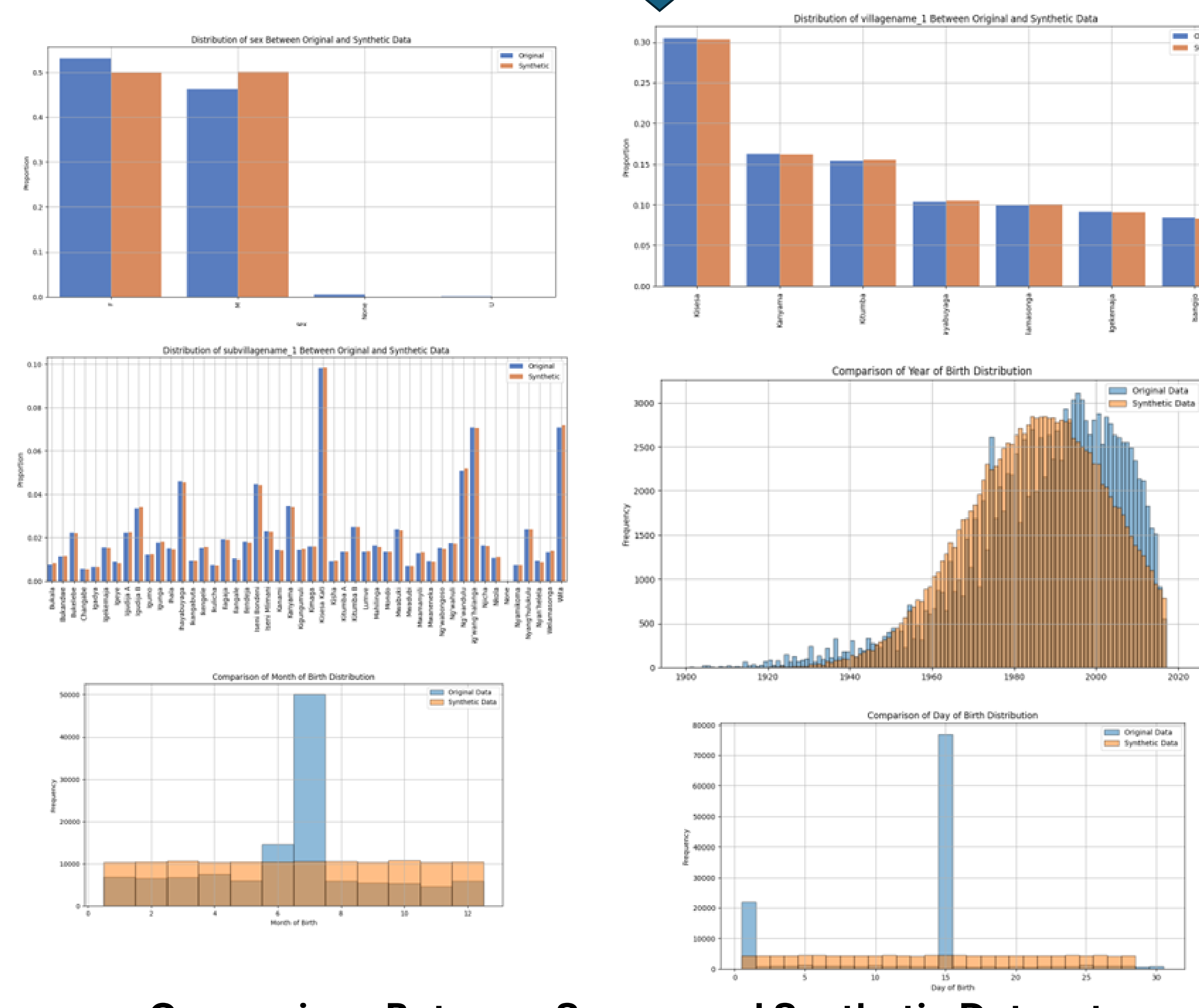



Kisesa ward
Mwanza region
Magu district
Map of Tanzania showing Mwanza Region, Magu District and Kisesa Ward

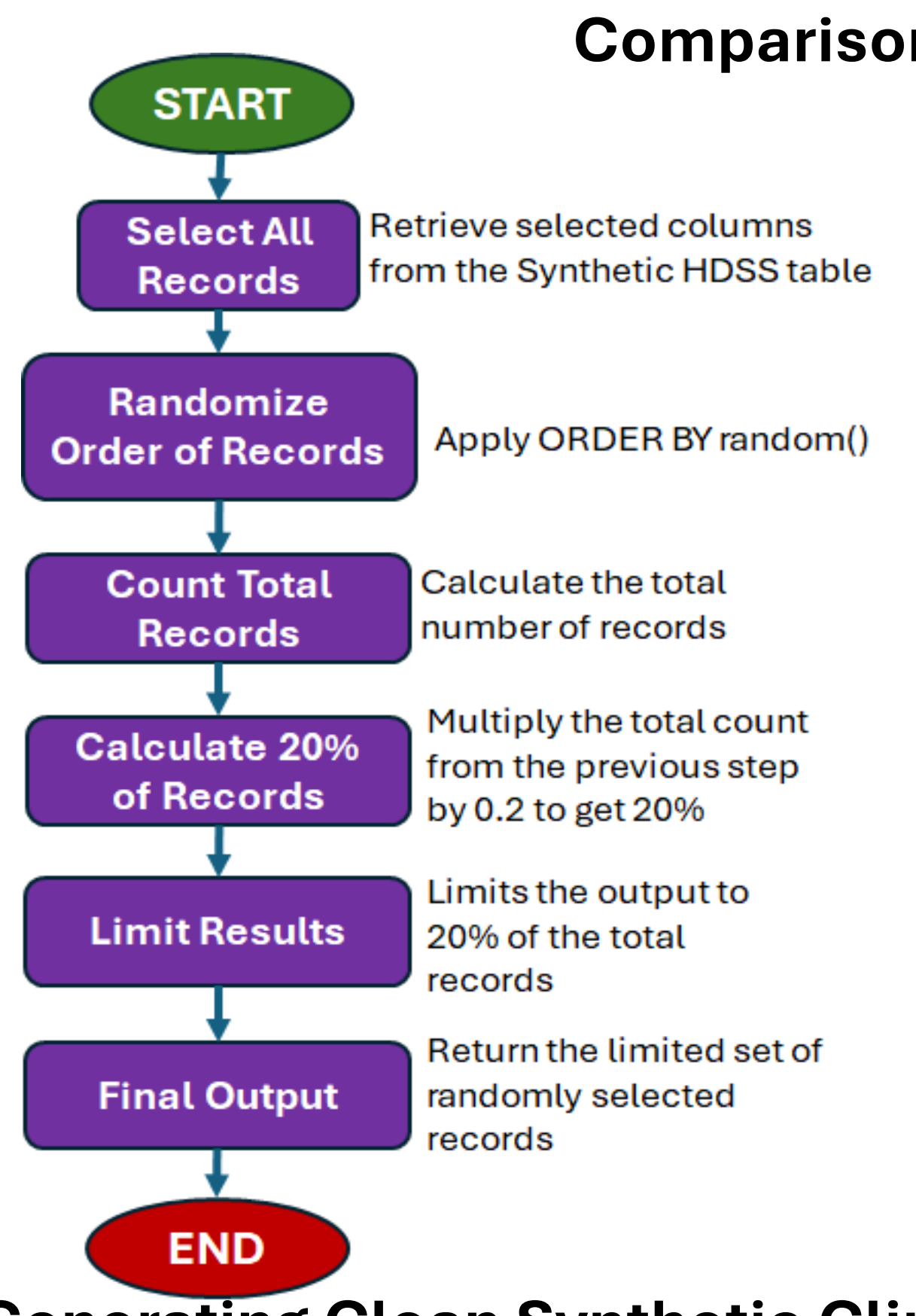
The Kisesa Health and Demographic Surveillance System (Kisesa HDSS) in North-Western Tanzania.


Source Features
Extract from Source PostgreSQL Database
Source Dataset
Extract Key Features
DoB
Gender
Names
Villages
Generate Normal Distribution for DoB
Generate Random Distribution of Name Specific Genders
Generate Random Distribution of Names
Generate Random Distribution of Villages
Write Data to Database
Generate Synthetic Data
Compare and Visualize

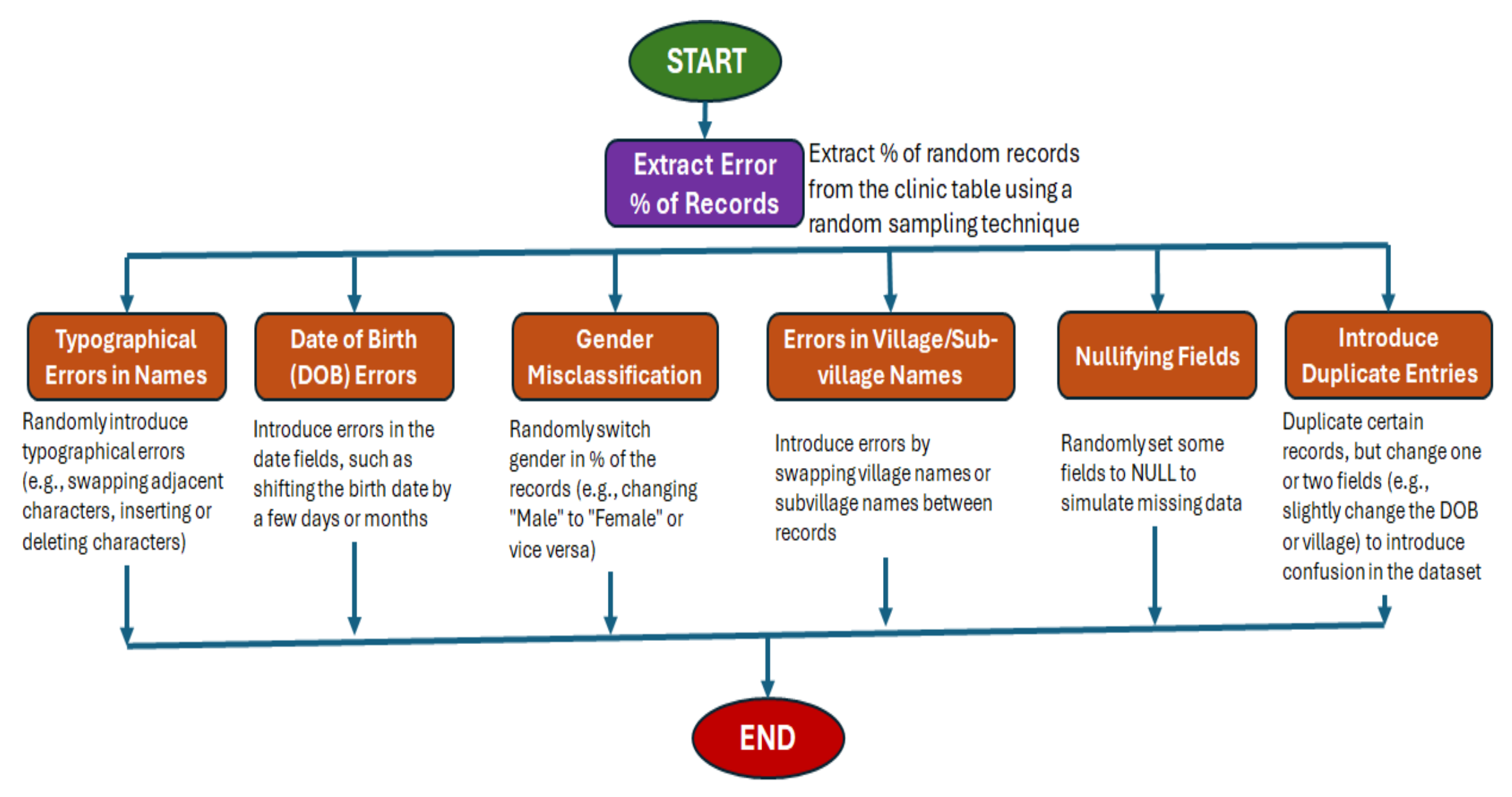
**Generation of Synthetic Dataset**



**Comparison Between Source and Synthetic Datasets**


START
Select All Records
Retrieve selected columns from the Synthetic HDSS table
Randomize Order of Records
Apply ORDER BY random()
Count Total Records
Calculate the total number of records
Calculate 20% of Records
Multiply the total count from the previous step by 0.2 to get 20%
Limit Results
Limits the output to 20% of the total records
Final Output
Return the limited set of randomly selected records
END

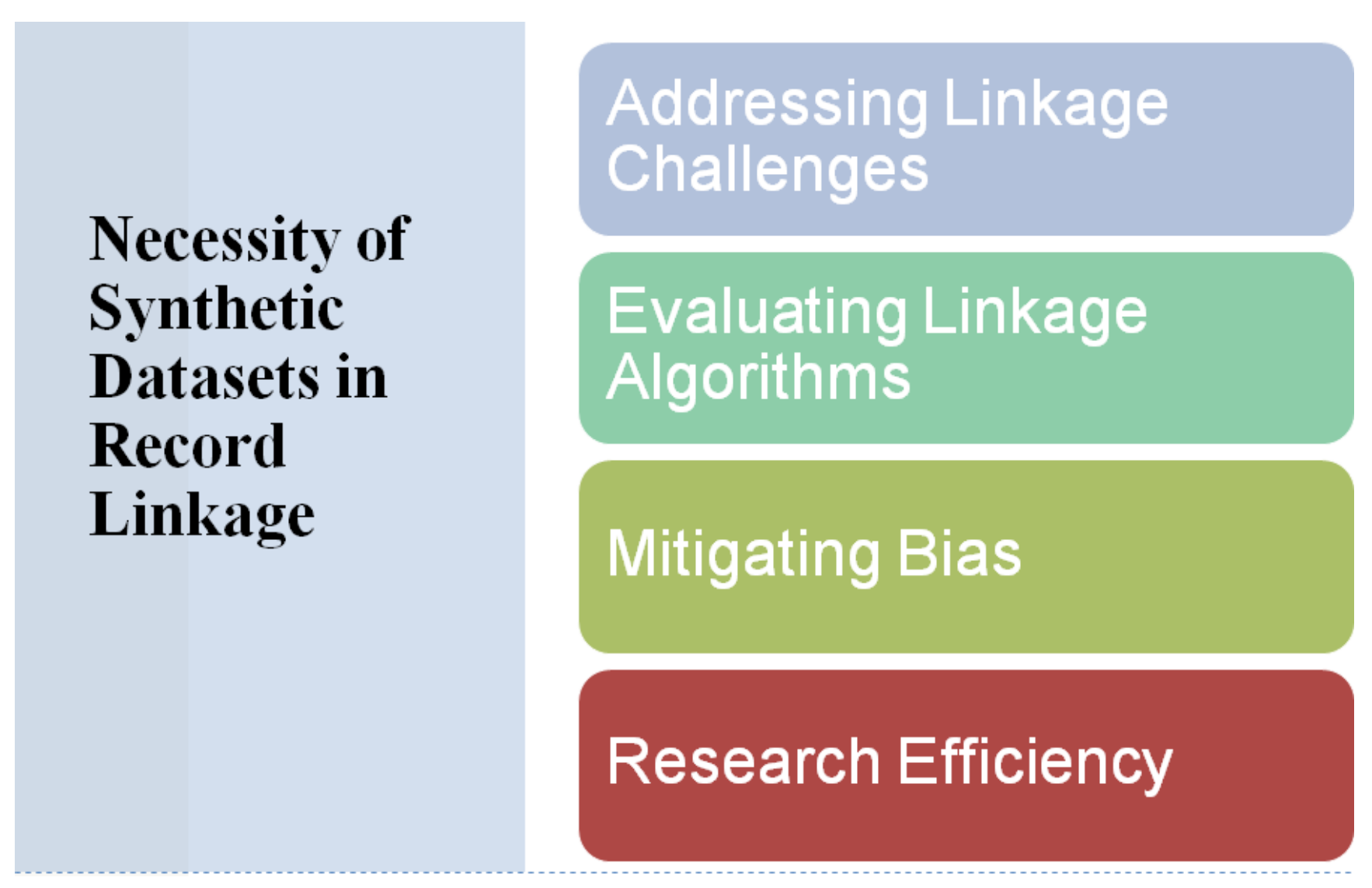
**Generating Clean Synthetic Clinic Data**


START
Extract Error % of Records
Extract % of random records from the clinic table using a random sampling technique
Typographical Errors in Names
Randomly introduce typographical errors (e.g., swapping adjacent characters, inserting or deleting characters)
Date of Birth (DOB) Errors
Introduce errors in the date fields, such as shifting the birth date by a few days or months
Gender Misclassification
Randomly switch gender in % of the records (e.g., changing "Male" to "Female" or vice versa)
Errors in Village/Sub-village Names
Introduce errors by swapping village names or subvillage names between records
Nullifying Fields
Randomly set some fields to NULL to simulate missing data
Introduce Duplicate Entries
Duplicate certain records, but change one or two fields (e.g., slightly change the DOB or village) to introduce confusion in the dataset
END

**Generating Forced Error Synthetic Clinic Data**

## Record Linkage

Record linkage is the process of connecting records from different datasets that refer to the same individual or entity. It helps integrate data for a comprehensive view, improving analysis in research, healthcare, and administrative applications.


Necessity of Synthetic Datasets in Record Linkage
Addressing Linkage Challenges
Evaluating Linkage Algorithms
Mitigating Bias
Research Efficiency

**Supervised ML** for record linkage, the model is trained on labeled data, where matched and unmatched pairs are known, allowing it to learn patterns for future predictions.

**Unsupervised ML**, no labeled data is provided, and the model groups records based on similarities (e.g., clustering) without prior knowledge of which records match.

Use case: **Unsupervised machine learning** (KMeans clustering) for record linkage (RL), matching records from two datasets without predefined labels using Python programming.

**Step 1: Load Data from PostgreSQL** Establish a connection to a PostgreSQL database and load data from two tables (HDSS and CLINIC)

The *SQLAlchemy* package is used to create a connection to the PostgreSQL database.
Two data frames (hdss_df and clinic_df) are loaded using SQL queries. These contain individual-level data from a health and demographic surveillance system (HDSS) and a clinic, respectively
Error handling catches any database connection issues (OperationalError) or errors while loading data (SQLAlchemyError).

**Step 2: Data Preprocessing** Prepare the data for further processing by combining the HDSS and clinic datasets into one data frame and handling missing values

Add a source column to each data frame (hdss or clinic) to keep track of the origin of the records. Concatenate the two data frames vertically into a single combined_df. Handle any missing values by filling them with an empty string

**Step 3: Feature Engineering** Convert categorical and string features into a numeric format to make the data suitable for clustering.

Date of Birth (dob): Converted to a UNIX timestamp (integer format representing seconds since January 1, 1970). Gender (sex): Mapped to binary values (1 for male, 0 for female), and missing values are filled with -1. Village and Subvillage Names: Encoded as categorical codes (each category is assigned a unique integer value). First and Last Names: Encoded as categorical codes.

**Step 4: Feature Engineering** Standardize the numeric features to make them comparable in scale for clustering.

StandardScaler from sklearn is used to scale all features (i.e., mean-centered and unit variance), ensuring that features with larger ranges do not dominate clustering.

**Step 5: Clustering Using KMeans** Apply KMeans clustering to group individuals based on the scaled features.

KMeans clustering is performed with 2 clusters (assuming two groups: matched and unmatched individuals).
The resulting cluster labels (0 or 1) are assigned to the combined_df in a new column cluster.

**Step 6: Evaluate the Clusters** Efficiently identify potential record matches between HDSS and clinic records using fuzzy string matching, and evaluate clustering performance.

The function are_strings_similar uses rapidfuzz to compare two strings based on a similarity threshold (80 in this case).The function is_match checks if both first names, last names, and dates of birth match between two records. The Parallel and delayed functions from joblib enable parallel processing to speed up comparisons between the HDSS and clinic datasets. A loop iterates over HDSS records and compares them with clinic records to identify matches. Results are stored in true_labels.

**Step 7: Silhouette Score for Clustering Evaluation** Evaluate the quality of clustering using the silhouette score.

The silhouette score measures how well the clustering algorithm has separated the data. A higher score indicates better-defined clusters. The silhouette score is calculated for the clustered data and printed.

**Step 8: Visualizing KMeans Clustering with PCA** Visualize the clusters using Principal Component Analysis (PCA) to reduce the feature dimensions to 2D

PCA reduces the high-dimensional data (scaled features) to two components for visualization. A scatter plot shows the clusters with color-coded cluster labels, making it easier to interpret how well the clustering has separated the records.

**Step 9: Generate Mapping CSV Using the 'id' Variable** Create a CSV file mapping matched id values between HDSS and clinic records

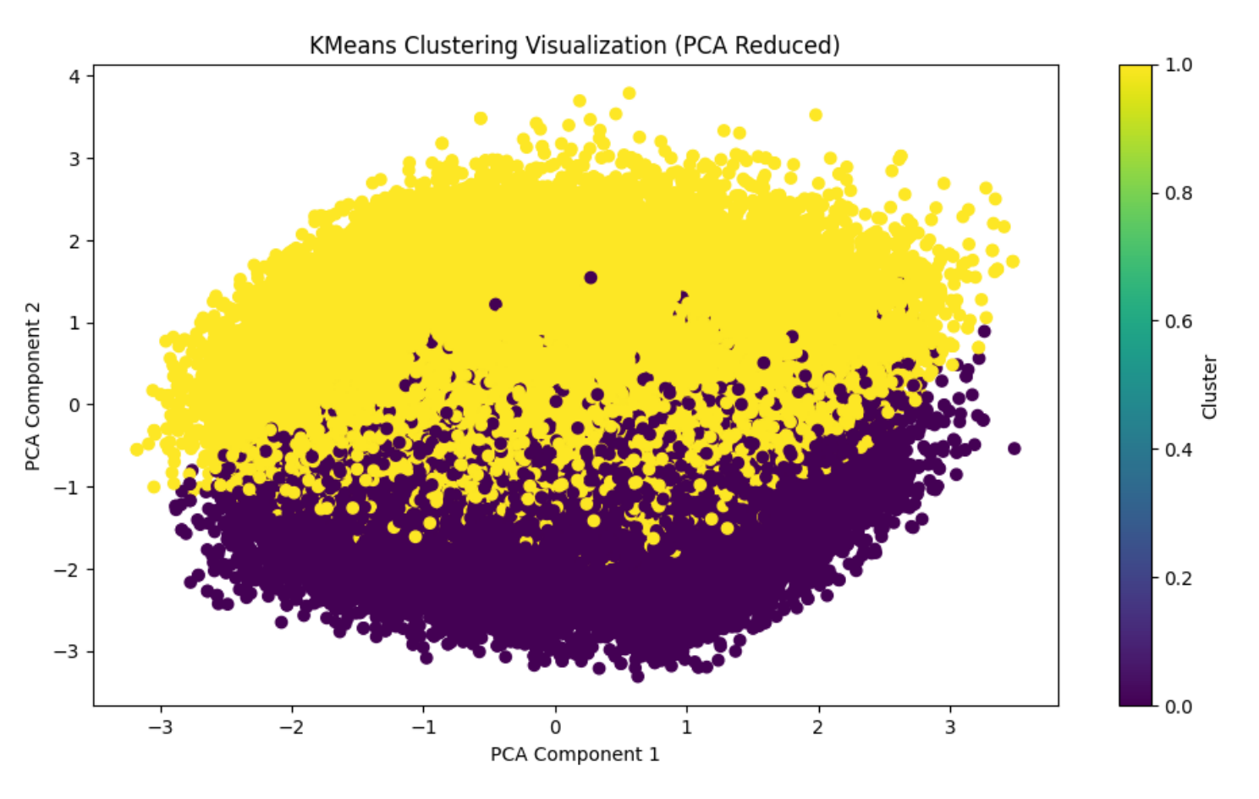
The rows where the KMeans clustering predicted a "match" (i.e., cluster_mapped equals 'match') are filtered. Ensure that HDSS and clinic records have corresponding id values. Handle any mismatch in the number of matches by trimming the larger dataset to avoid index errors. The final mapping of hdss_id to clinic_id is saved to a CSV file.

**Step 10: Close the Database Connection** Properly close the database connection after completing the operations.

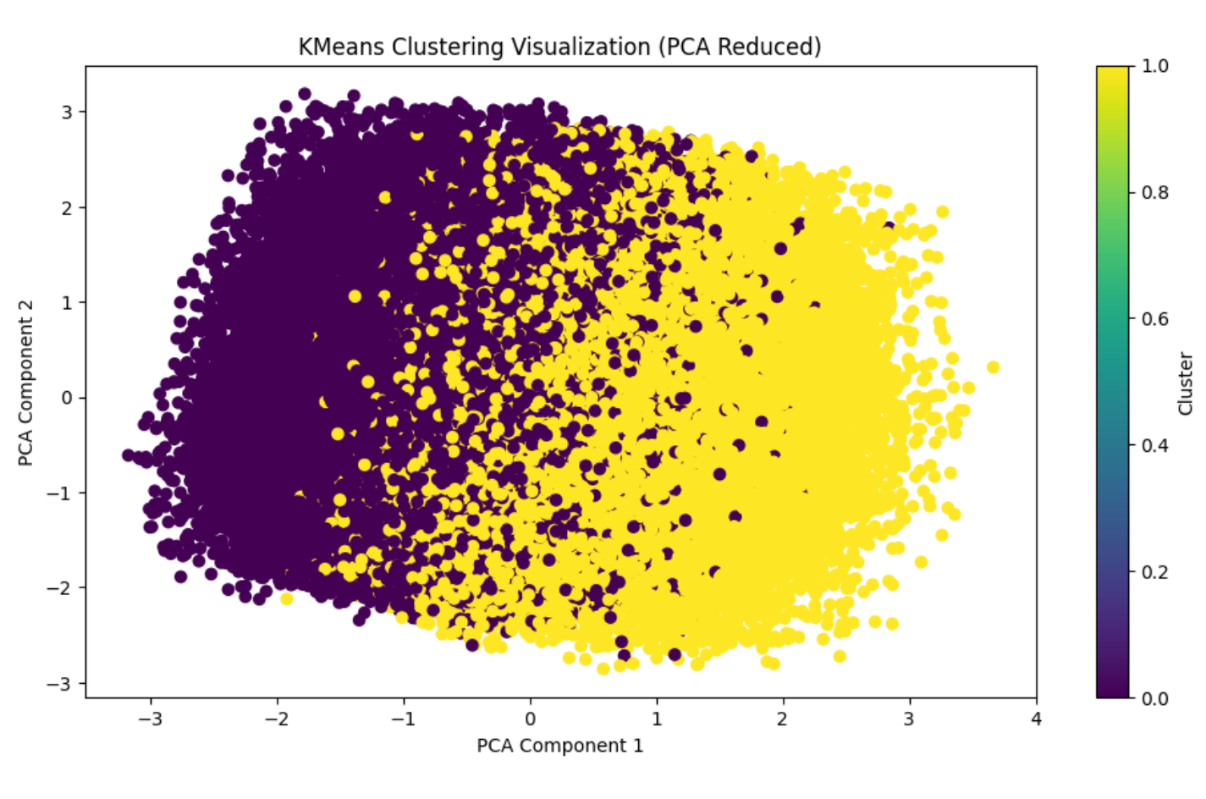
The engine.dispose() function is called to close the connection to the PostgreSQL database, freeing up resources.

**HDSS Dataset: 125852 individuals.** **Clinic Dataset: 25170 individuals**

Record Linkage Cases with Different Levels of Errors: Scenarios where data records from the two sources were compared and matched despite varying degrees of errors or inconsistencies. These errors could arise from spelling variations, missing information, or formatting differences in names, dates of birth, or addresses. By examining record linkage cases with different levels of such induced errors, we can assess the robustness of matching algorithms, ensuring they can handle real-world data imperfections effectively



Record Linkage with less errors in datasets
Matches: 18188



Record Linkage with more errors in datasets
Matches: 6307

**Conclusion:** The comparison between the two datasets highlights the effect of increased data errors on record linkage performance. In the cleaner dataset, the algorithm showed better precision but missed many matches. With the noisier dataset, precision for non-matches dropped drastically, and recall for matches decreased, indicating more difficulty in handling errors. The silhouette score also slightly worsened (from 0.17 to 0.16), reflecting less clear clustering. Additionally, the mismatch between HDSS and clinic records increased, showing the challenge of linking data with higher errors. This underscores the need for better preprocessing or robust methods to handle noisy data.

Acknowledgements:
The Kisesa Health & Demographic Surveillance System | Analysing Longitudinal Population-based HIV/AIDS data on Africa (ALPHA Network) | Implementation Network for Sharing Population Information from Research Entities (INSPIRE Network)