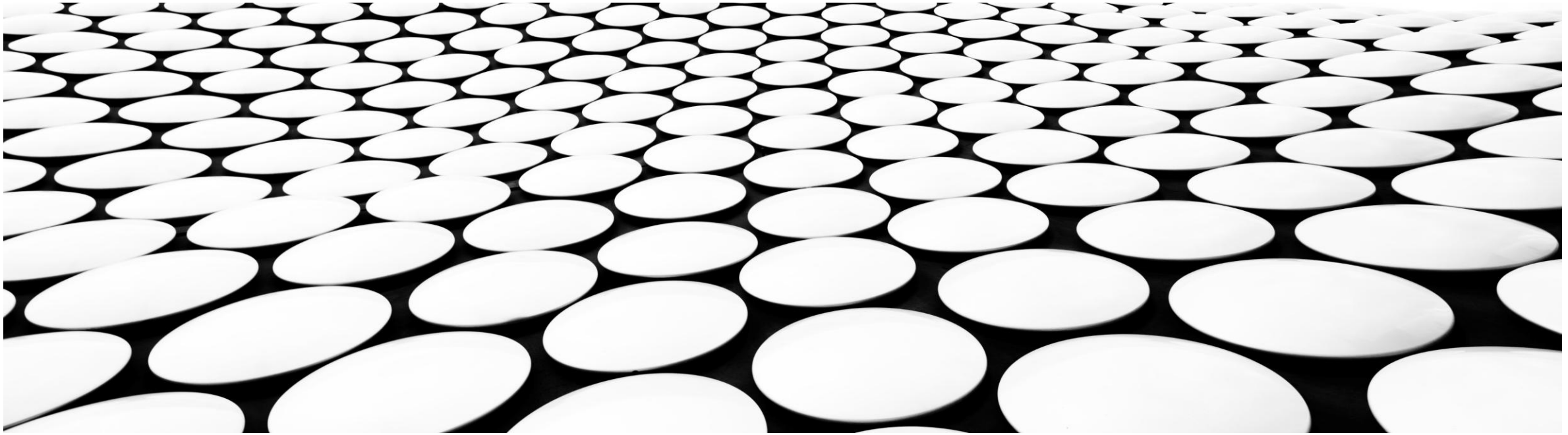
Data Science Without Borders

CREATING SYNTHETIC DATASETS TO OVERCOME DATA GOVERNANCE CHALLENGES

TATHAGATA BHATTACHARJEE



DATA GOVERNANCE

- **Data Governance** is the **framework of policies, processes, standards, and controls** that ensure the proper management, security, and use of data within an organization.
- It focuses on **data quality, privacy, compliance, and accessibility**, ensuring that data is reliable and used ethically.
- Key Components
  - Data Quality – Ensuring accuracy, completeness, and consistency of data
  - Data Privacy & Security – Protecting sensitive information and complying with regulations
  - Data Compliance – Adhering to legal and industry standards (e.g., GDPR, HIPAA)
  - Data Access & Control – Defining who can access and modify data
  - Data Lifecycle Management – Managing data from creation to deletion

IMPORTANCE OF DATA GOVERNANCE

- **Ensures Compliance** – Meets legal and ethical obligations
- **Protects Privacy** – Prevents unauthorized access to sensitive data
- **Improves Data Quality** – Reduces errors and inconsistencies
- **Enhances Decision-Making** – Reliable data leads to better insights
- **Facilitates Data Sharing** – Enables secure and ethical data exchange

- **Challenge:** While data governance strengthens data security, it often limits data accessibility, making synthetic data a crucial solution for privacy-preserving analytics.

SYNTHETIC DATA

- **Definition:** Synthetic data is artificially generated data that mimics real-world data while preserving statistical properties.

- **Types of Synthetic Data:**
  - **Fully Synthetic:** No real data points used
  - **Partially Synthetic:** Some real and some synthetic elements
  - **Differentially Private Synthetic Data:** Preserves privacy with mathematical guarantees

WHY SYNTHETIC DATA?

- **Why is Data Governance a Challenge?**
  - Growing concerns over **privacy** and **security** in data sharing
  - **Regulations** like GDPR, HIPAA, and national policies restrict data access
  - **Limited availability** of real-world datasets for research and innovation
  - Ethical concerns in using sensitive personal data

- **How Can Synthetic Data Help?**
  - **Artificially generated** but statistically accurate data
  - Retains **patterns, distributions, and relationships** of original data
  - Enables **safe data sharing** without exposing sensitive information

THE NEED FOR SYNTHETIC DATA

- **Challenges with Real Data:**
  - **Privacy Risks:** Exposure of Personally Identifiable Information (PII)
  - **Regulatory Restrictions:** Legal barriers to data sharing
  - **Data Scarcity:** Limited access to high-quality datasets
  - **Bias & Imbalance:** Real data may not represent all cases

- **How Synthetic Data Solves This:**
  - **Privacy-Preserving:** No real individuals data is used
  - **Regulation-Compliant:** Can be shared freely without legal concerns
  - **Accessible & Scalable:** Generated on demand for various scenarios
  - **Bias Reduction:** Can be balanced and diversified

METHODS OF GENERATING SYNTHETIC DATA

- ## Rule-Based Methods
  - Uses predefined rules and logic to generate synthetic data.
  - Example: Random sampling from distributions, shuffling real data.
- ## Statistical Methods
  - Generates synthetic data based on statistical properties of real data.
  - Example: Gaussian Mixture Models (GMM),
- ## Machine Learning-Based Methods
  - Uses ML models to learn patterns and generate synthetic data.
  - Example: Decision trees, regression models.
- ## Deep Learning-Based Methods
  - Uses neural networks to generate high-fidelity synthetic data.
  - Example: **Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs).**
- ## Differential Privacy-Based Methods
  - Adds controlled noise to ensure privacy while maintaining data utility.
  - Example: DP-Synthetic Data Generation (DP-SDG).

GENERATIVE AI

- A subset of artificial intelligence (AI)
- Focuses on creating new content, like text, images, audio, video, or even code, based on patterns and data it has been trained on
- Unlike traditional AI systems that are designed for specific tasks (e.g., classification or prediction), Generative AI models are capable of producing original outputs that mimic human creativity
- Characteristics of Generative AI
  - Creates New Content: It generates new data (e.g., text, images, music) rather than just analyzing or classifying existing data.
  - Learns from Data: It is trained on large datasets to understand patterns, structures, and relationships within the data
  - Uses Advanced Models:
    - Generative Adversarial Networks (GANs): Two neural networks (a generator and a discriminator) work together to create realistic outputs.
    - Variational Autoencoders (VAEs): Models that learn to encode and decode data to generate new samples.
    - Transformer-based Models: Models like GPT (Generative Pre-trained Transformer) that excel in generating text data.

EXAMPLES OF GENERATIVE AI

- Text Generation:
  - Tools like ChatGPT, GPT-4, and Bard, Copilot that generate human-like text for conversations, essays, or code.
  - Example: Writing a story, summarizing an article, or answering questions.
- Image Generation:
  - Tools like DALL·E, MidJourney, and Stable Diffusion create images from text descriptions.
  - Example: Generating a picture of "a futuristic city on Mars.“
- Audio Generation:
  - Tools like ElevenLabs or VALL-E generate realistic speech or music.
  - Example: Creating a voiceover or composing a song.
- Video Generation:
  - Tools like Synthesia or Runway ML generate videos from text or images.
  - Example: Creating a marketing video with AI-generated actors.
- Code Generation:
  - Tools like GitHub Copilot or Codex generate code based on natural language prompts.
  - Example: Writing a Python script for data analysis.

HOW GENERATIVE AI WORKS?

- Training:
  - The model is trained on a large dataset (e.g., text, images, or audio).
  - It learns patterns, relationships, and structures in the data.
- Generation:
  - Once trained, the model can generate new content by predicting the next word, pixel, or note based on the input it receives.
- Fine-Tuning:
  - Models can be fine-tuned for specific tasks or domains to improve their performance.

GENERATIVE ADVERSARIAL NETWORK (GAN)

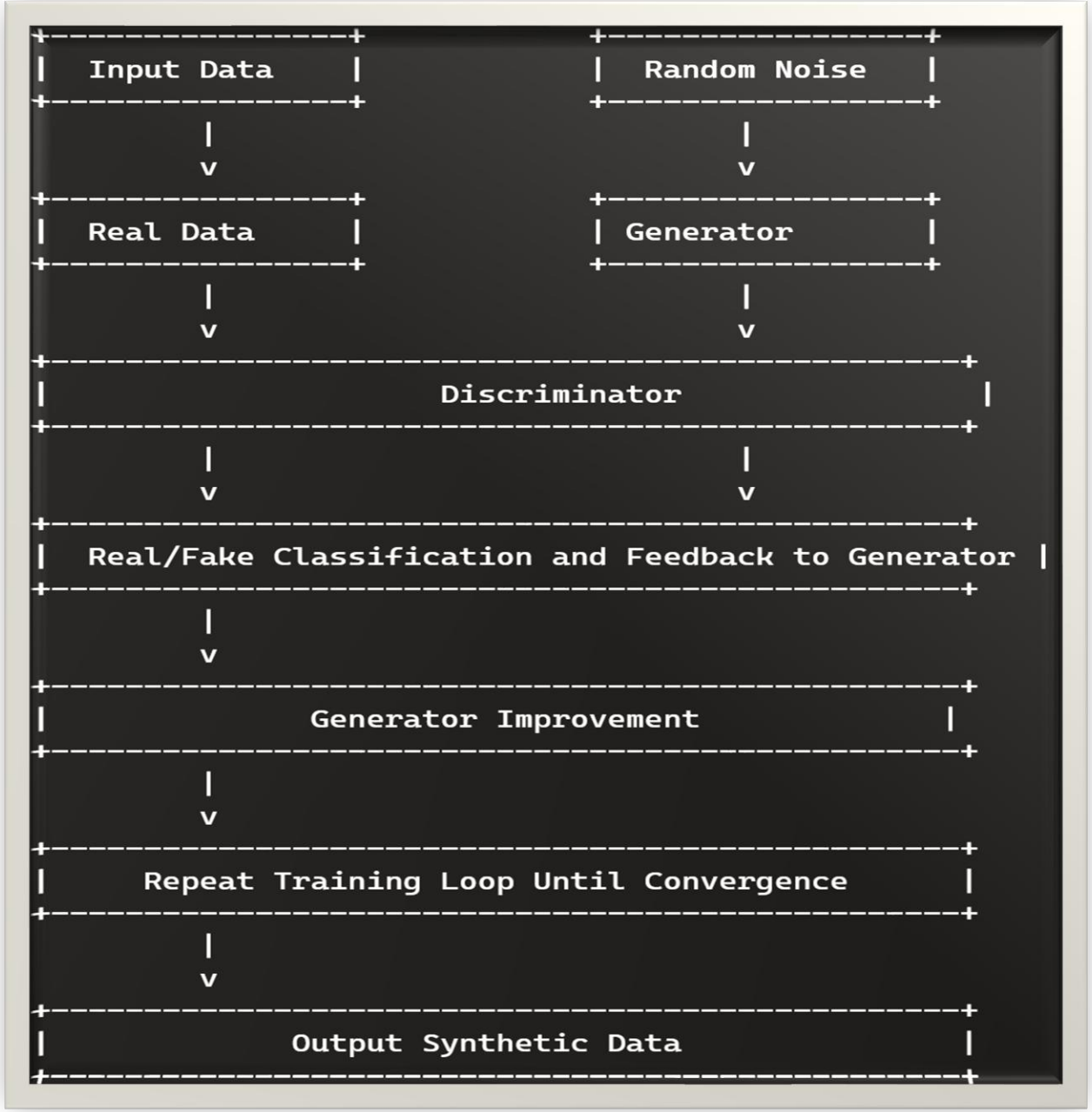
- A deep learning model designed to generate synthetic data that resembles real data.
- Ian Goodfellow introduced it in 2014, and it is widely used in fields such as image generation, data augmentation, and synthetic data creation.
- GAN consists of two competing neural networks:
  - Generator (G) – Creates fake data from random noise.
  - Discriminator (D) – Evaluates and distinguishes between real and fake data.
- Adversarial – meaning: involving or characterized by conflict or opposition.

GAN PROCESS FOR GENERATING SYNTHETIC TABULAR DATA

- **Input Data:** Begin with the real tabular data
- **Generator Initialization:** Initialize the generator network, which will generate synthetic data samples.
- **Discriminator Initialization**: Initialize the discriminator network, which will evaluate the authenticity of the samples produced by the generator.
- Training Loop:
  - Step 1: **Generator Generates Data**: The generator creates synthetic data samples based on random noise input.
  - Step 2: **Discriminator Evaluates Data**: The discriminator evaluates both real and synthetic data samples to determine their authenticity.
  - Step 3: **Feedback to Generator**: The discriminator provides feedback on the synthetic data, indicating whether it was classified as real or fake.
  - Step 4: **Generator Improvement**: The generator updates its parameters based on the feedback to produce more realistic synthetic data.
  - Step 5: **Repeat**: Repeat steps 1 to 4 for multiple iterations until the generator produces high-quality synthetic data that the discriminator cannot easily distinguish from real data.
- **Output Synthetic Data**: Once the training is complete, the generator produces synthetic tabular data that closely resembles the real data.

GAN FLOW DIAGRAM

- The flow of data and the interaction between the **generator** and **discriminator** during the GAN training process for generating synthetic tabular data.
- Each iteration of the training loop helps the generator produce more realistic synthetic data, ultimately resulting in high-quality synthetic samples.

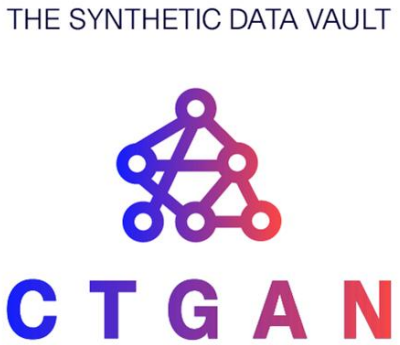


WHY USE GANS FOR SYNTHETIC DATA?

- **Preserves Data Privacy** – No need to expose real sensitive data.
- **Captures Complex Data Distributions** – Generates realistic samples.
- **Adaptable to Different Domains** – Works for images, text, and tabular data.

CTGAN (CONDITIONAL TABULAR GENERATIVE ADVERSARIAL NETWORK)

- CTGAN is an advanced type of GAN designed specifically for generating realistic tabular data, including both continuous and categorical variables.
- The MIT Data introduced it to AI Lab and is part of the Synthetic Data Vault (SDV) project.
- Unlike traditional GANs, which struggle with tabular data due to complex distributions, CTGAN handles imbalanced categorical variables and multimodal distributions effectively.
- Why Use CTGAN for Synthetic Data Generation?
  - **Handles Mixed Data Types** – Works well with numerical and categorical data.
  - **Solves Data Imbalance Issues** – Resamples minority classes effectively.
  - **Captures Complex Data Distributions** – Uses a mode-specific normalization technique.
  - **Preserves Relationships Between Features** – Generates realistic synthetic data.



CTGAN COMPONENTS

- **Generator (G)** – Learns patterns in real tabular data and generates synthetic data.
- **Discriminator (D)** – Evaluates the authenticity of synthetic data and improves differentiation.
- **Mode-Specific Normalization** – Transforms continuous variables into a structured format.
- **Training-by-Sampling** – Ensures categorical data is well-represented.

CTGAN PROCESS

- Preprocessing & Encoding
  - Continuous values are transformed using mode-specific normalization to handle skewed distributions.
  - Categorical values are converted into numerical representations.
- Training-by-Sampling (Two Networks Competing)
  - Balances categorical data by sampling underrepresented classes more frequently
  - Generator (G)
    - Takes random noise + a sampled condition (e.g., a specific category).
    - Generates synthetic tabular data based on learned patterns.
  - Discriminator (D)
    - Takes real and synthetic data.
    - Classifies whether a given sample is real or fake.
- Optimization & Learning
  - The Generator improves by minimizing the Discriminator's ability to detect fake data.
  - The Discriminator learns to better classify real vs. fake data.
- Conditional Training
  - To prevent mode collapse, the Generator conditions its output on real categories sampled from the training data.
  - This ensures that all categorical values are represented correctly.
- Generate Synthetic Data
  - Once trained, CTGAN generates high-quality synthetic data that retains the statistical properties of the original dataset.
  - Categorical and numerical values retain realistic distributions.

GAN VS CTGAN

- Traditional GANs struggle with tabular data
- **Imbalanced categorical variables** – Some categories may dominate, leading to poor diversity in generated data.
- **Multi-modal distributions in continuous variables** – Real-world numerical data is often **non-Gaussian** and has complex patterns.
- **Sparse Data Challenges** – Missing values or rare categories make training difficult.

- CTGAN overcomes limitations
- **Mode-specific Normalization** – Captures multi-modal distributions in continuous variables.
- **Conditional Sampling** – Ensures that all categories appear in synthetic data.
- **Training by Sampling Rare Categories More Often** – Balances data generation.

ADVANTAGES OF CTGAN

- **Captures Complex Tabular Data Patterns** – Handles multi-modal and imbalanced distributions.
- **Preserves Data Privacy** – No real data exposure, making it useful for sensitive datasets.
- **Works with Missing & Rare Data** – Unlike traditional GANs, it adapts to sparse datasets.
- **Useful for Machine Learning Tasks** – Synthetic data can be used to train models without bias.

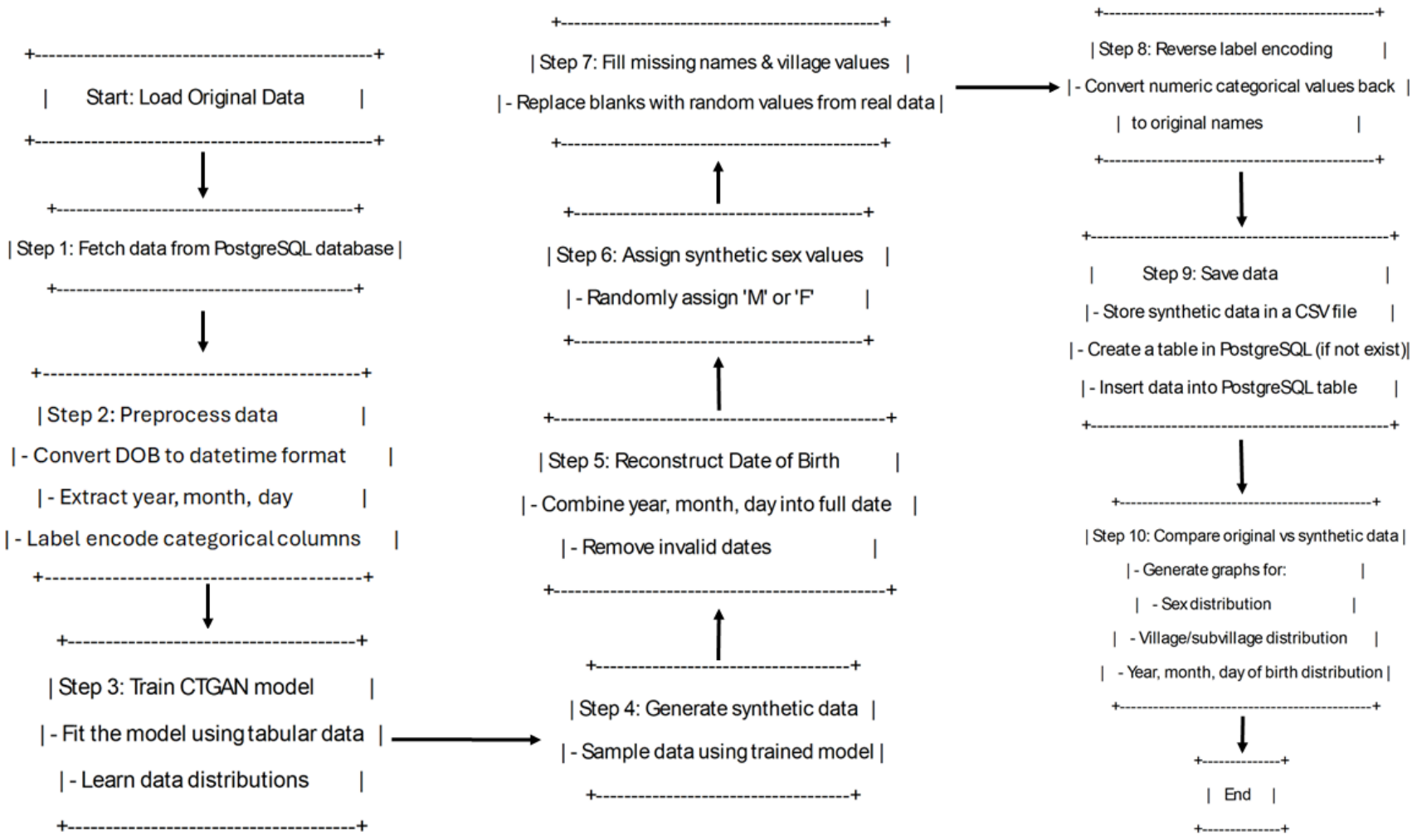
GENERATING SYNTHETIC DATA USING CTGAN

Understanding the Process, Code, and Implementation

OBJECTIVE OF THIS IMPLEMENTATION

- **Goal:** Generate synthetic data using CTGAN
- **Approach:**
  - Use real data as input.
  - Train CTGAN to learn patterns.
  - Generate synthetic samples.
  - Store the results in CSV & database.
  - Compare distributions using visualizations.
- GitHub: https://tinyurl.com/5dbzb6s3

FLOWCHART REPRESENTATION



STEP 1 – LOADING THE ORIGINAL DATA

- Source: PostgreSQL Database
- Query Used: Selecting relevant columns (idlong, firstname, lastname, sex, dob, villagename, subvillagename).
- Ensuring Data Quality:
  - Ordering by idlong for consistency.
  - Handling missing values in the next step.

```
engine = create_engine('postgresql://postgres:password1234@localhost:5432/PhD')
query = """
    SELECT idlong, firstname, lastname, sex, dob,
    villagename_1 AS villagename,
    subvillagename_1 AS subvillagename
    FROM long_to_wide.pirl_rentch_hdss_r29_rdlinkageinputdatafinali_wide
                ORDER BY idlong;
"""
original_data = pd.read_sql(query, engine)

# Display shape of original data
print(f'Original Data Shape: {original_data.shape[0]} rows, {original_data.shape[1]}
columns')
```

STEP 2 – DATA PREPROCESSING

- Convert dob to DateTime format.
- Extract date components: year, month, day.
- Handle categorical columns:
  - Encode firstname, lastname, sex, villagename, subvillagename using Label Encoding.
- Why Label Encoding?
  - CTGAN needs numeric inputs for training.

```
original_data['dob'] = pd.to_datetime(original_data['dob'], errors='coerce')
original_data.dropna(subset=['dob'], inplace=True)

# Extract year, month, and day from 'dob' and drop 'dob' column
original_data['year_of_birth'] = original_data['dob'].dt.year
original_data['month_of_birth'] = original_data['dob'].dt.month
original_data['day_of_birth'] = original_data['dob'].dt.day
original_data.drop(columns=['dob'], inplace=True)

# Handle categorical columns and create mappings for original values
categorical_cols = ['firstname', 'lastname', 'sex', 'villagename', 'subvillagename']
original_value_maps = {}  # Store the mappings
print("\n Handling Categorical columns done")

for col in categorical_cols:
    le = LabelEncoder()
    original_data[col] = le.fit_transform(original_data[col].astype(str))  # Ensure
conversion to str
    original_value_maps[col] = le  # Store the LabelEncoder for inverse
transformation
```

STEP 3 – TRAINING THE CTGAN MODEL

- What is CTGAN?
  - A deep learning model designed for tabular synthetic data generation.
- Training Details:
  - 100 epochs (iterations).
  - Uses Generative Adversarial Networks (GANs).
- Expected Output:
  - A trained model that can generate synthetic samples.

```
ctgan = CTGAN(epochs=100, verbose=True)
ctgan.fit(original_data)
```

An epoch is when all the training data is used at once and is defined as the total number of iterations of all the training data in one cycle for training the machine learning model.
Another way to define an epoch is the number of passes a training dataset takes around an algorithm.

STEP 4 – GENERATING SYNTHETIC DATA

- Using the trained CTGAN model to generate synthetic records.
- Output: Same structure as original data, but values are synthetic.

```
synthetic_data = ctgan.sample(len(original_data))
```

STEP 5 – RECONSTRUCTING DATE OF BIRTH

- Combining year, month, and day into a valid date.
- Handling invalid dates (NaT values are removed).

```
synthetic_data['date_of_birth'] = pd.to_datetime({
    'year': synthetic_data['year_of_birth'],
    'month': synthetic_data['month_of_birth'],
    'day': synthetic_data['day_of_birth']
}, errors='coerce')

# Drop rows with NaT in date_of_birth if present
synthetic_data.dropna(subset=['date_of_birth'], inplace=True)
```

STEP 6 & 7 – HANDLING SEX VALUES

```
synthetic_sex_numeric = np.random.choice([1, 2],
size=len(synthetic_data))  # 1 for 'M', 2 for 'F'
synthetic_data['sex'] = synthetic_sex_numeric  # Store
numeric values temporarily
```

- Generate random values (1 or 2) representing M or F
- Convert numeric values back to string (M, F).
- **Why?**
  - Ensures a balanced gender distribution in synthetic data.

```
synthetic_data['sex'] = synthetic_data['sex'].replace({1: 'M', 2: 'F'})
```

STEP 8 – HANDLING MISSING NAMES AND VILLAGES

- Issue: Some synthetic values might be blank.
- Solution:
  - Fill missing firstname, lastname, villagename,

  and subvillagename with random values from original data.
- Why?
  - Ensures no blank entries in synthetic dataset.

```
for col in ['firstname', 'lastname', 'villagename',
'subvillagename']:
    random_values =
original_data[col].dropna().unique()
    blank_or_missing = (synthetic_data[col].isna()) |
(synthetic_data[col] == '')
    blank_count = blank_or_missing.sum()
    if blank_count > 0:
        synthetic_data.loc[blank_or_missing, col] =
np.random.choice(random_values, size=blank_count)
```

STEP 9 – REVERSING LABEL ENCODING

- Transform numeric categorical values back to their original names.
- Why is this necessary?
  - Ensures readability & interpretability.
- Final Shape of Data:
  - Matches the original dataset format.

```
for col in ['firstname', 'lastname', 'villagename', 'subvillagename']:
    le = original_value_maps[col]
    value_mapping = {i: le.inverse_transform([i])[0] for i in
range(len(le.classes_))}
    synthetic_data[col] =
synthetic_data[col].map(value_mapping).fillna('')

# Display shape of synthetic data
print(f'Synthetic Data Shape: {synthetic_data.shape[0]} rows,
{synthetic_data.shape[1]} columns')
```

STEP 10 & 11 – STORING THE DATA

- Step 10: Save to CSV file (synthetic_GAN_data_without_date_normalization.csv).

```
synthetic_data.to_csv('synthetic_GAN_data_without_date_normalization.csv', index=False)
```

- Step 11: Save to PostgreSQL Database:
  - Table Name: synthetic.synthetic_without_date_normalization_v1
  - Columns: idlong, firstname, lastname, sex, date_of_birth, villagename, subvillagename.

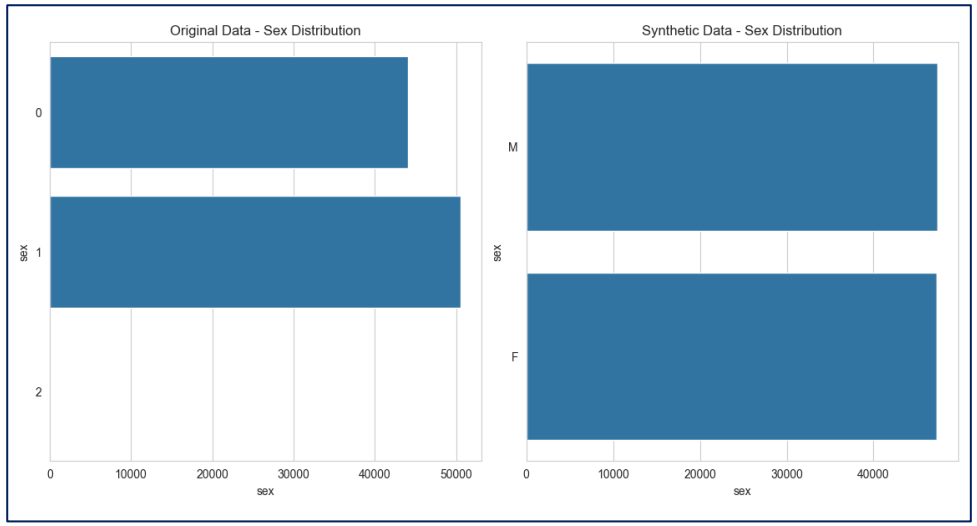
```
synthetic_data.to_sql('synthetic_without_date_normalization_v1', con=engine, schema='synthetic', if_exists='replace', index=False)
```

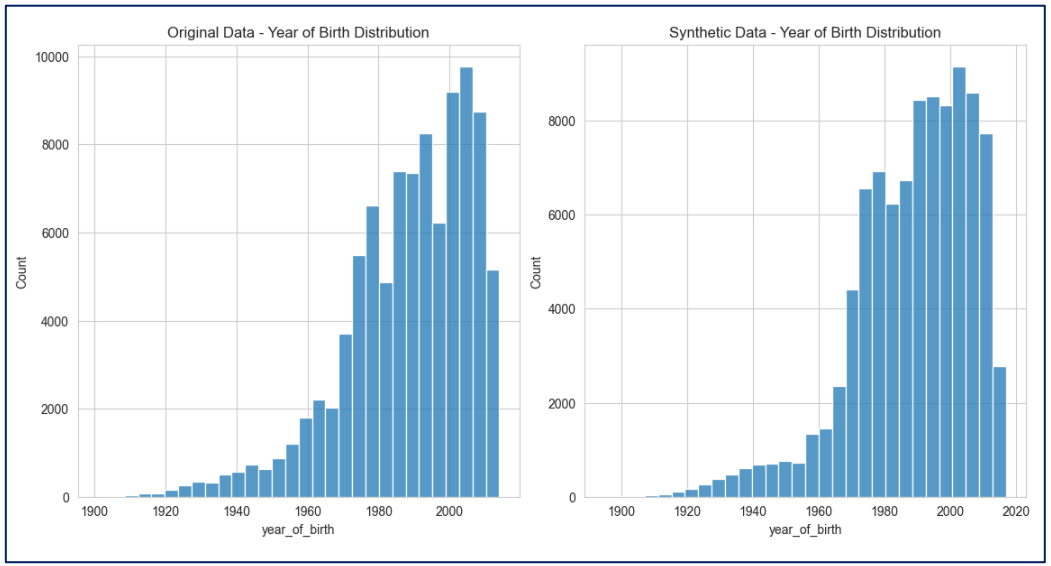
STEP 12 – DATA COMPARISON USING GRAPHS

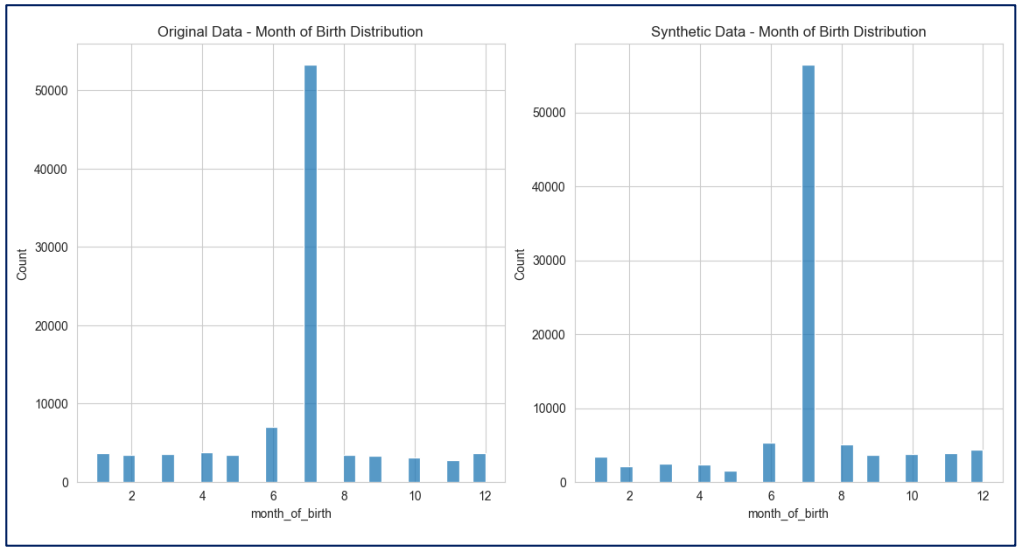
- Comparing Original vs Synthetic Data:
  - Sex Distribution: Bar plot
  - Village Name Distribution: Bar plot
  - Sub-Village Name Distribution: Bar plot
  - Year of Birth Distribution: Histogram
  - Month of Birth Distribution: Histogram
  - Day of Birth Distribution: Histogram
- Visualization Tools:
  - matplotlib & seaborn

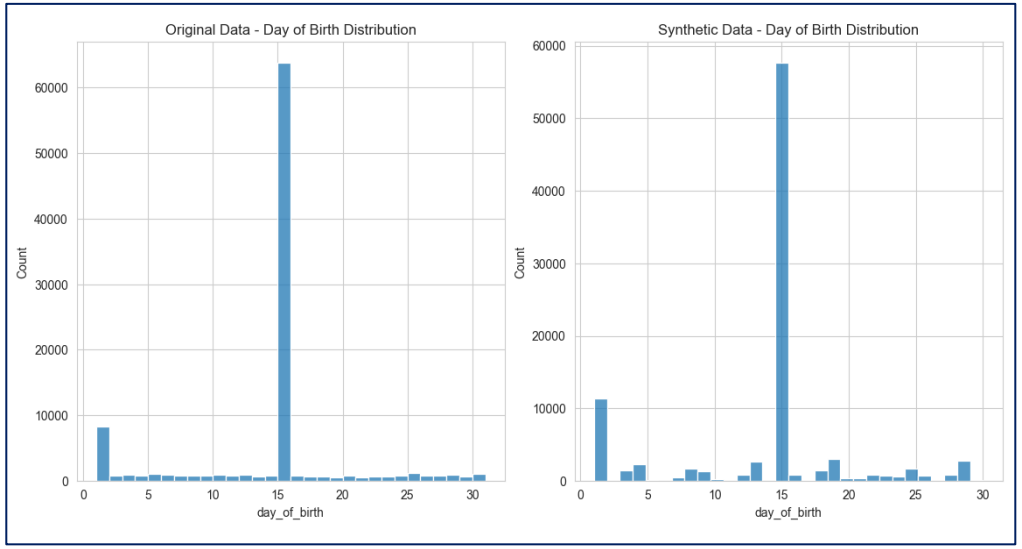
```
sns.set_style("whitegrid")

# Define a function to plot comparisons
def plot_comparison(original, synthetic, column, title, kind='bar'):
    fig, axes = plt.subplots(1, 2, figsize=(12, 5))

    # Original Data
    sns.histplot(original[column], kde=False, ax=axes[0], bins=30) if kind == 'hist' else sns.countplot(
        y=original[column], ax=axes[0])
    axes[0].set_title(f'Original Data - {title}')
    axes[0].set_xlabel(column)

    # Synthetic Data
    sns.histplot(synthetic[column], kde=False, ax=axes[1], bins=30) if kind == 'hist' else sns.countplot(
        y=synthetic[column], ax=axes[1])
    axes[1].set_title(f'Synthetic Data - {title}')
    axes[1].set_xlabel(column)

    plt.tight_layout()
    plt.show()

# Compare categorical variables (sex, villagename, subvillagename)
plot_comparison(original_data, synthetic_data, 'sex', 'Sex Distribution')
plot_comparison(original_data, synthetic_data, 'villagename', 'Village Name Distribution')
plot_comparison(original_data, synthetic_data, 'subvillagename', 'Sub-Village Name Distribution')

# Compare numerical variables (year_of_birth, month_of_birth, day_of_birth)
plot_comparison(original_data, synthetic_data, 'year_of_birth', 'Year of Birth Distribution', kind='hist')
plot_comparison(original_data, synthetic_data, 'month_of_birth', 'Month of Birth Distribution', kind='hist')
plot_comparison(original_data, synthetic_data, 'day_of_birth', 'Day of Birth Distribution', kind='hist')
```

KISESA HDSS VS SYNTHETIC DATA









GENERATING SYNTHETIC DATA WITH WELL-DISTRIBUTED DATE OF BIRTH USING CTGAN

Understanding the Process, Code, and Implementation

KEY MODIFICATIONS:

- Standardizing Date of Birth:
  - Convert the date into a normalized range (e.g., min-max scaling or z-score normalization).
  - Ensure that generated synthetic dates are evenly spread across the range.
- Ensuring a Distribution:
  - Use a Gaussian or uniform distribution when generating synthetic birthdates.
  - Avoid clustering around specific years or days

MODIFICATIONS IN THE CODE – WELL DISTRIBUTION IN MONTH & DAY

- Previous Code:
  - CTGAN-generated month_of_birth and day_of_birth had spikes (e.g., many records had January 1st).
- Modified Code:
  - Ensures a well-distributed synthetic DOB by generating months (1-12) and days (1-28) uniformly.

Previous Code:

```
# Extract year, month, and day from synthetic data
synthetic_data['year_of_birth'] = synthetic_data['date_of_birth'].dt.year
synthetic_data['month_of_birth'] = synthetic_data['date_of_birth'].dt.month
synthetic_data['day_of_birth'] = synthetic_data['date_of_birth'].dt.day
```

Modified Code:

```
# Assign realistic year from inverse-transformed dob_numeric
synthetic_data['year_of_birth'] = synthetic_data['date_of_birth'].dt.year
# Assign well-distributed months and days
synthetic_data['month_of_birth'] = np.random.randint(1, 13, size=len(synthetic_data))   # 1 to 12
synthetic_data['day_of_birth'] = np.random.randint(1, 29, size=len(synthetic_data))  # 1 to 28 (to avoid leap year issues)
```

MODIFICATIONS IN THE CODE – IMPROVED SYNTHETIC DATA ACCURACY

- Previous Code:
  - Used **100 epochs (iterations)**, leading to **less realistic** synthetic data
- Modified Code:
  - Increased epochs to 300 for better model convergence and synthetic data quality.

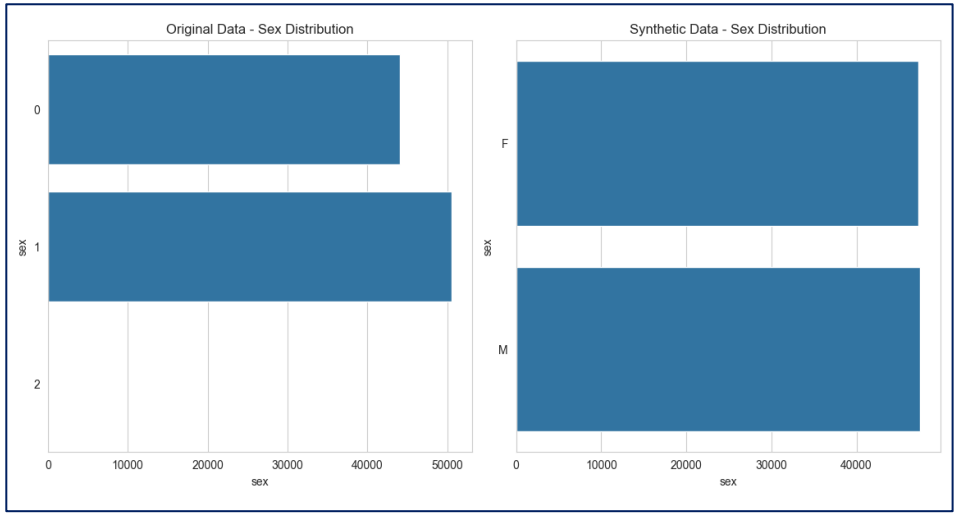
Previous code:

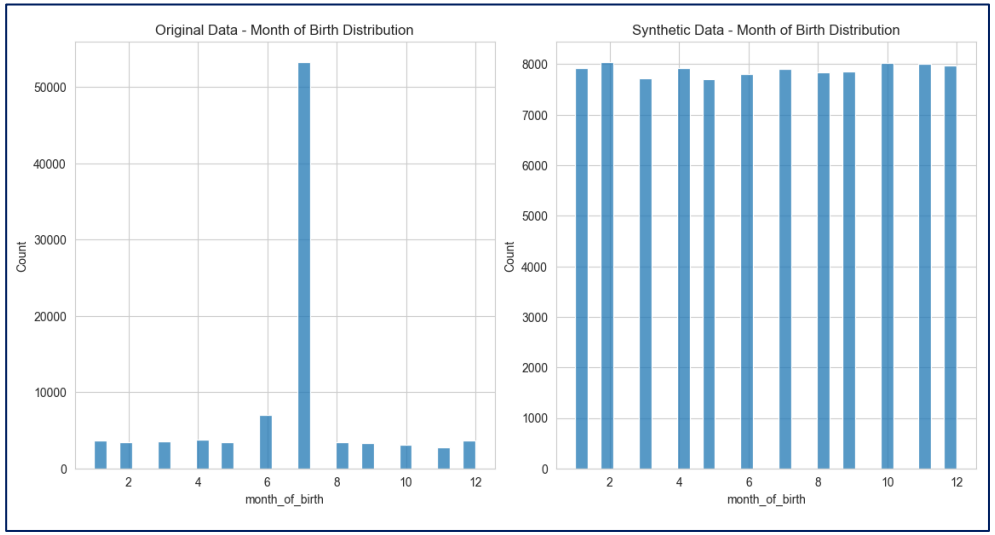
```
ctgan = CTGAN(epochs=100, verbose=True)
```

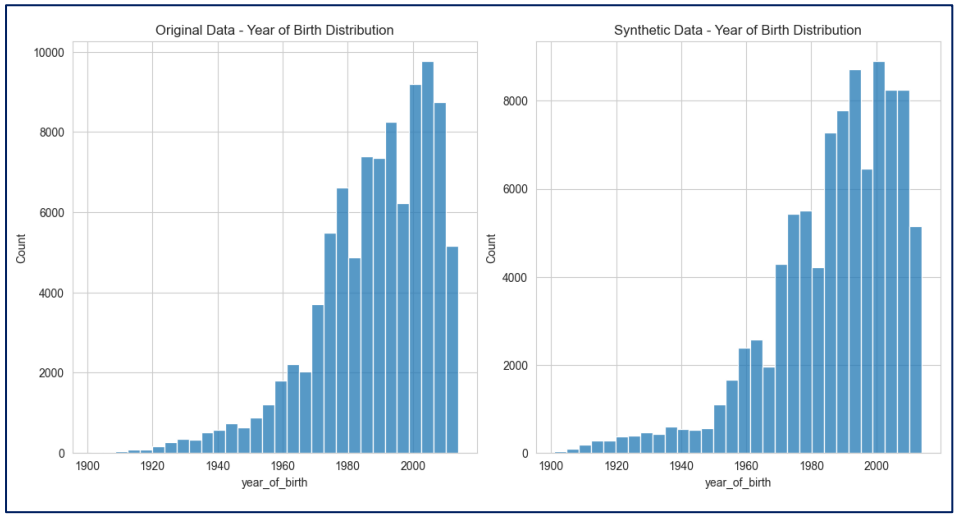
Modified code:

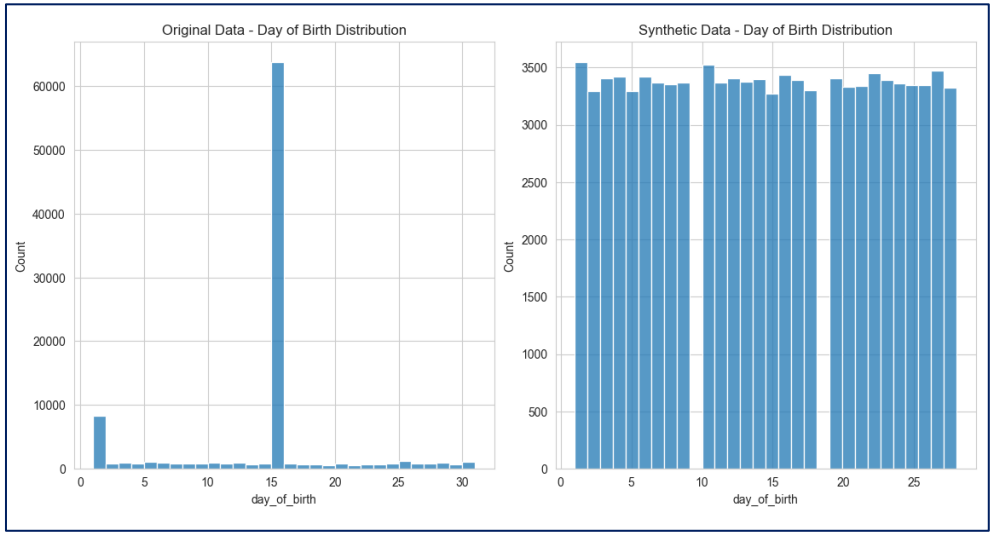
```
ctgan = CTGAN(epochs=300, verbose=True)
```

KISESA HDSS VS SYNTHETIC DATA (WELL DISTRIBUTED DOB)









Tathagata Bhattacharjee

tathagata.bhattacharjee@lshtm.ac.uk | https://www.lshtm.ac.uk/aboutus/people/bhattacharjee.tathagata | https://orcid.org/0000-0001-9437-0894 | www.linkedin.com/in/tathagatabhattacharjee