This code is to generate synthetic data from a source data using ML Methods

This code uses the CTGAN (Conditional Tabular Generative Adversarial Network)

The sklearn PyPI package is deprecated, install scikit-learn

Install ctgan from Git: pip install git+https://github.com/sdv-dev/CTGAN.git

This code is written by Tathagata Bhattacharjee

In [ ]:
```
import pandas as pd
import numpy as np
from sqlalchemy import create_engine, text
from sklearn.preprocessing import LabelEncoder
import ctgan
from ctgan import CTGAN
import matplotlib.pyplot as plt
import seaborn as sns
```

##

STEP 1: Load Original Data

Establish connection to the PostgreSQL database

##

In [ ]:
```
engine = create_engine('postgresql://postgres:password1234@localhost:5432/PhD')

# Query to fetch relevant columns from the source table
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)

# Print the shape of the loaded dataset
print(f'Original Data Shape: {original_data.shape[0]} rows, {original_data.shape[1]
```

##

STEP 2: Preprocess Data

Convert date of birth column to datetime format and remove invalid dates

##

In [ ]:
```
original_data['dob'] = pd.to_datetime(original_data['dob'], errors='coerce')
original_data.dropna(subset=['dob'], inplace=True)  # Remove rows where DOB is NaT
print("\n Date conversion and NaN removal done")

# Extract year, month, and day components from '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

# Remove the original 'dob' column as it is no longer needed
original_data.drop(columns=['dob'], inplace=True)

# Handle categorical columns by encoding them
categorical_cols = ['firstname', 'lastname', 'sex', 'villagename', 'subvillagename'
original_value_maps = {}  # Store mappings for inverse transformation
print("\n Handling Categorical columns done")

# Label encode categorical columns to numerical format for CTGAN
for col in categorical_cols:
    le = LabelEncoder()
    original_data[col] = le.fit_transform(original_data[col].astype(str))  # Ensure
    original_value_maps[col] = le  # Store the LabelEncoder for future use
print("\n Label Encoding done")
```

##

STEP 3: Fit the CTGAN Model

Initialize CTGAN model with specified epochs

```
##########################################
```

In [ ]:
```
ctgan = CTGAN(epochs=100, verbose=True)

# Train the model using the transformed original dataset
ctgan.fit(original_data)
print("\n Step 3 to fit the CTGAN model completed")
```

```
##########################################
```

STEP 4: Generate Synthetic Data

Generate a synthetic dataset with the same number of records as the original dataset

```
##########################################
```

In [ ]:
```
synthetic_data = ctgan.sample(len(original_data))
```

```
##########################################
```

STEP 5: Generate synthetic dates using year, month, and day components

```
##########################################
```

In [ ]:
```
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 invalid synthetic dates (NaT)
synthetic_data.dropna(subset=['date_of_birth'], inplace=True)
```

```
##########################################
```

STEP 6: Assign 'M' or 'F' to synthetic sex values

Assign random binary values representing 'M' and 'F' to maintain realism

##

In [ ]:
```
synthetic_sex_numeric = np.random.choice([1, 2], size=len(synthetic_data))  # 1 for
synthetic_data['sex'] = synthetic_sex_numeric  # Temporarily store numeric values
```

##

STEP 7: Convert numeric sex back to 'M' and 'F'

##

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

##

STEP 8: Fill missing or blank names and villages with random values from the original dataset

##

In [ ]:
```
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,
```

##

STEP 9: Convert numeric categorical values back to original strings

##

In [ ]:
```
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('')

# Print the shape of the synthetic dataset
print(f'Synthetic Data Shape: {synthetic_data.shape[0]} rows, {synthetic_data.shape
```

##

STEP 10: Save the synthetic dataset to a CSV file

##

In [ ]:
```
synthetic_data.to_csv('synthetic_GAN_data_without_date_normalization.csv', index=Fa
print("\n Synthetic data saved successfully.")
```

##

STEP 11: Store Synthetic Data in PostgreSQL

Step 11a: Create the table if it does not exist

##

In [ ]:
```
create_table_query = """
CREATE TABLE IF NOT EXISTS synthetic.synthetic_without_date_normalization_v1 (
    idlong BIGINT,
    firstname TEXT,
    lastname TEXT,
    sex TEXT,
    date_of_birth DATE,
    villagename TEXT,
    subvillagename TEXT
);
"""

with engine.connect() as connection:
    connection.execute(text(create_table_query))  # Create table
```

##

Step 11b: Insert synthetic data into the table

##

In [ ]:
```
synthetic_data.to_sql('synthetic_without_date_normalization_v1', con=engine, schema
print("\n Synthetic data saved successfully to the database.")
```

##

STEP 12: Generate Graphical Comparisons

##

In [ ]:
```
sns.set_style("whitegrid")

def plot_comparison(original, synthetic, column, title, kind='bar'):
    fig, axes = plt.subplots(1, 2, figsize=(12, 5))
    sns.histplot(original[column], kde=False, ax=axes[0], bins=30) if kind == 'hist
    axes[0].set_title(f'Original Data - {title}')
    sns.histplot(synthetic[column], kde=False, ax=axes[1], bins=30) if kind == 'his
    axes[1].set_title(f'Synthetic Data - {title}')
    plt.tight_layout()
    plt.show()

plot_comparison(original_data, synthetic_data, 'sex', 'Sex Distribution')
plot_comparison(original_data, synthetic_data, 'villagename', 'Village Name Distrib
plot_comparison(original_data, synthetic_data, 'subvillagename', 'Sub-Village Name
plot_comparison(original_data, synthetic_data, 'year_of_birth', 'Year of Birth Dist
plot_comparison(original_data, synthetic_data, 'month_of_birth', 'Month of Birth Di
plot_comparison(original_data, synthetic_data, 'day_of_birth', 'Day of Birth Distri
print("\n Step 12: Graphical comparisons generated successfully.")
```

##