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

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

This code generates the synthetic data with a well distributed DoB

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, MinMaxScaler
from ctgan import CTGAN
import matplotlib.pyplot as plt
import seaborn as sns
```

STEP 1: Load Original Data

In [ ]:
```
engine = create_engine('postgresql://postgres:password1234@localhost:5432/synthetic
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 RANDOM();
"""
original_data = pd.read_sql(query, engine)
print(f'Original Data Shape: {original_data.shape[0]} rows, {original_data.shape[1]}
```

STEP 2: Preprocess Data

In [ ]:
```
original_data['dob'] = pd.to_datetime(original_data['dob'], errors='coerce')
original_data.dropna(subset=['dob'], inplace=True)  # Remove invalid DOBs

# Convert DOB to numeric days since 1970-01-01
epoch = pd.Timestamp("1970-01-01")
original_data['dob_numeric'] = (original_data['dob'] - epoch).dt.days

# Normalize DOB using MinMaxScaler to avoid spikes
min_max_scaler = MinMaxScaler()
original_data['dob_numeric'] = min_max_scaler.fit_transform(original_data[['dob_num

# Extract year, month, and day separately
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

# Drop original DOB column after extraction
original_data.drop(columns=['dob'], inplace=True)

# Handle categorical columns using Label Encoding
categorical_cols = ['firstname', 'lastname', 'sex', 'villagename', 'subvillagename'
original_value_maps = {}

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

STEP 3: Fit the CTGAN Model

In [ ]:
```
ctgan = CTGAN(epochs=300, verbose=True)  # Increased epochs for better learning
ctgan.fit(original_data)
```

STEP 4: Generate Synthetic Data

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

STEP 5: Convert Normalized DOB Back to Date Format

In [ ]:
```
synthetic_data['dob_numeric'] = np.clip(synthetic_data['dob_numeric'], 0, 1)
synthetic_data['date_of_birth'] = epoch + pd.to_timedelta(
    min_max_scaler.inverse_transform(synthetic_data[['dob_numeric']]).flatten(), un
)
synthetic_data.drop(columns=['dob_numeric'], inplace=True)  # Remove numeric DOB af

# Extract and distribute synthetic birth dates properly
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

# Ensure months and days are well-distributed (avoid spikes)
synthetic_data['month_of_birth'] = np.random.choice(range(1, 13), size=len(syntheti
synthetic_data['day_of_birth'] = np.random.choice(range(1, 29), size=len(synthetic_
```

STEP 6: Convert Numeric Sex Back to 'M' and 'F'

In [ ]:
```
synthetic_data['sex'] = np.random.choice(['M', 'F'], size=len(synthetic_data))
```

STEP 7: Fill Missing or Blank Names/Villages with Random Values

---

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 8: Re-encode Numeric Columns Back to Original String Values

---

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(f'Synthetic Data Shape: {synthetic_data.shape[0]} rows, {synthetic_data.shape
```

STEP 9: Store the Synthetic Data to a CSV File

In [ ]:
```
synthetic_data.to_csv('synthetic_GAN_data_with_distributed_dob_v1.csv', index=False
print("\n Synthetic data saved successfully.")
```

STEP 10: Store the Synthetic Data to a Table

In [ ]:
```
create_table_query = """
CREATE TABLE IF NOT EXISTS synthetic.synthetic_with_distributed_dob_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 if not exists

synthetic_data.to_sql('synthetic_with_distributed_dob_v1', con=engine, schema='synt
print("\n Synthetic data saved successfully to the database.")
```

STEP 11: Compare Data Graphically

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

# Function to plot distributions
def plot_comparison(original, synthetic, column, title, kind='bar'):
    if column not in original or column not in synthetic:
        print(f"Skipping {column}, as it is missing in one dataset.")
        return

    fig, axes = plt.subplots(1, 2, figsize=(12, 5))

    if kind == 'hist':
        sns.histplot(original[column], kde=False, ax=axes[0], bins=30)
        sns.histplot(synthetic[column], kde=False, ax=axes[1], bins=30)
    else:
        sns.countplot(y=original[column], ax=axes[0])
        sns.countplot(y=synthetic[column], ax=axes[1])

    axes[0].set_title(f'Original Data - {title}')
```

```
    axes[0].set_xlabel(column)
    axes[1].set_title(f'Synthetic Data - {title}')
    axes[1].set_xlabel(column)

    plt.tight_layout()
    plt.show()

# Compare distributions
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 11: Graphical comparisons generated successfully.")
```

In [ ]:

```
# ------------------------------------------------
# END OF CODE
# ------------------------------------------------
```