```
# ================================================================
# Script: Synthetic Data DSWB Training.py
# Purpose: Generate synthetic data from a PostgreSQL source using CTGAN
# Author: Tathagata Bhattacharjee
# Dependencies: CTGAN (via GitHub), scikit-learn, pandas, SQLAlchemy, matplotlib, seaborn
# Install ctgan from Git: pip install git+https://github.com/sdv-dev/CTGAN.git
# Note: Ensure all required Python packages are installed before running.
# ================================================================

# ----------------------------- #
# Import required dependencies #
# ----------------------------- #
import pandas as pd  # For data handling and manipulation
import numpy as np  # For numerical operations and sampling
from sqlalchemy import create_engine, text  # To connect and query a PostgreSQL database
from sklearn.preprocessing import LabelEncoder  # To encode categorical variables for modeling
from ctgan import CTGAN  # Conditional Tabular GAN for generating synthetic tabular data
import matplotlib.pyplot as plt  # (Optional) For visualization
import seaborn as sns  # (Optional) For enhanced plots
from datetime import timedelta  # For date calculations

# ---------------------------------------- #
# STEP 1: Load Source Data from PostgreSQL  #
# ---------------------------------------- #
print("\n##########################################")
print("\nSTEP 2: STEP 1: Load Source Data\n")

# Create connection to PostgreSQL database
engine = create_engine('postgresql://postgres:password1234@localhost:5432/dswb_training')

# SQL query to extract relevant fields from the table
query = """
    SELECT id, first_name, last_name, sex, dob, village_name, occupation,
    covid_19_first_vacc_date, age_on_first_vacc, vacc_manufacturer
    FROM synthetic_dswb_training.source_dataset_fake
    ORDER BY id;
"""

# Load query result into a pandas DataFrame
source_data = pd.read_sql(query, engine)

# Print number of rows and columns for verification
print(f'Source Data Shape: {source_data.shape[0]} rows, {source_data.shape[1]} columns')

# ----------------------------- #
# STEP 2: Preprocess the dataset #
# ----------------------------- #
print("\n##########################################")
print("\nSTEP 2: Preprocess Data\n")

# Convert 'dob' and 'covid_19_first_vacc_date' to datetime format; drop rows with invalid dates
source_data['dob'] = pd.to_datetime(source_data['dob'], errors='coerce')
source_data.dropna(subset=['dob'], inplace=True)

source_data['covid_19_first_vacc_date'] = pd.to_datetime(source_data['covid_19_first_vacc_date'], errors='coerce')
source_data.dropna(subset=['covid_19_first_vacc_date'], inplace=True)

print("\n Date conversion for DoB and Vaccination Date and NaN removal done")

# Extract date components into separate columns (useful for CTGAN modeling)
source_data['year_of_birth'] = source_data['dob'].dt.year
source_data['month_of_birth'] = source_data['dob'].dt.month
source_data['day_of_birth'] = source_data['dob'].dt.day
source_data['year_of_vacc'] = source_data['covid_19_first_vacc_date'].dt.year
source_data['month_of_vacc'] = source_data['covid_19_first_vacc_date'].dt.month
source_data['day_of_vacc'] = source_data['covid_19_first_vacc_date'].dt.day
```

```
66
67  # Drop source datetime columns after extraction
68  source_data.drop(columns=['dob', 'covid_19_first_vacc_date'], inplace=True)
69
70  # List of categorical columns to encode
71  categorical_cols = ['id', 'first_name', 'last_name', 'sex', 'village_name',
    'occupation', 'vacc_manufacturer']
72  source_value_maps = {}  # Dictionary to store LabelEncoders for decoding later
73  print("\n Handling Categorical columns done")
74
75  # Apply Label Encoding on each categorical column
76  for col in categorical_cols:
77      le = LabelEncoder()
78      source_data[col] = le.fit_transform(source_data[col].astype(str))  # Encode as string
79      source_value_maps[col] = le  # Save encoder for later decoding
80  print("\n Label Encoding done")
81
82  # Define numerical columns for CTGAN (includes date parts and age)
83  numerical_cols = ['year_of_birth', 'month_of_birth', 'day_of_birth', 'year_of_vacc',
    'month_of_vacc', 'day_of_vacc', 'age_on_first_vacc']
84
85  # Store male/female first names from source data for name assignment later
86  male_names_source = source_data[source_data['sex'] == 1]['first_name'].unique()
87  female_names_source = source_data[source_data['sex'] == 0]['first_name'].unique()
88
89  # Store source sex distribution (used to validate synthetic output)
90  sex_distribution_source = source_data['sex'].value_counts(normalize=True)
91
92  # ------------------------------------ #
93  # STEP 3: Train the CTGAN model        #
94  # ------------------------------------ #
95  print("\n#########################################")
96  print("\nSTEP 3: Fit the CTGAN Model\n")
97
98  # Initialize CTGAN with specified training epochs
99  ctgan = CTGAN(epochs=1000, verbose=True)
100
101 # Train the CTGAN on preprocessed source data
102 ctgan.fit(source_data, discrete_columns=categorical_cols)
103 print("\n Step 3 to fit the CTGAN model completed")
104
105 # ---------------------------------------------- #
106 # STEP 4: Generate Synthetic Data from CTGAN     #
107 # ---------------------------------------------- #
108 print("\n#########################################")
109 print("\nSTEP 4: Generate Synthetic Data\n")
110
111 # Generate synthetic data of the same size as source
112 synthetic_data = ctgan.sample(len(source_data))
113
114 # Display diagnostic information
115 print(f'Synthetic Data Shape: {synthetic_data.shape[0]} rows, {synthetic_data.shape[1]}
    columns')
116 print("\nColumns in synthetic_data after sampling:")
117 print(synthetic_data.columns)
118 print("\nFirst few rows of synthetic_data after sampling with encoding:")
119 print(synthetic_data.head())
120 print("\nData types of synthetic_data after sampling:")
121 print(synthetic_data.dtypes)
122
123 # ------------------------------------------------------------------------ #
124 # STEP 5: Reconstruct dates and filter for adults (age >= 18 at vaccination)
125 # ------------------------------------------------------------------------ #
126 print("\n#########################################")
127 print("\nSTEP 5: Reconstructing dates and ensuring adult vaccination & sex-name
    consistency\n")
128
129 # Round date components to ensure valid integers, then convert to Int64 (nullable)
130 for col in ['year_of_birth', 'month_of_birth', 'day_of_birth', 'year_of_vacc',
```

```
    'month_of_vacc', 'day_of_vacc']:
        synthetic_data[col] = synthetic_data[col].round().astype('Int64')

    # Rebuild full date columns from components
    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')

    synthetic_data['date_of_vacc'] = pd.to_datetime({
        'year': synthetic_data['year_of_vacc'],
        'month': synthetic_data['month_of_vacc'],
        'day': synthetic_data['day_of_vacc']
    }, errors='coerce')

    # Compute age at vaccination in years
    synthetic_data['age_at_vaccination'] = (synthetic_data['date_of_vacc'] -
    synthetic_data['date_of_birth']).dt.days / 365.25

    # Iteratively re-sample non-adult records (<18) to ensure adult population
    adult_threshold = 18
    while True:
        non_adults = synthetic_data[synthetic_data['age_at_vaccination'] < adult_threshold]
        if non_adults.empty:
            break
        print(f"\nGenerated {len(non_adults)} non-adult records. Re-sampling...")

        resampled_non_adults = ctgan.sample(len(non_adults))
        for col in ['year_of_birth', 'month_of_birth', 'day_of_birth', 'year_of_vacc',
        'month_of_vacc', 'day_of_vacc']:
            resampled_non_adults[col] = resampled_non_adults[col].round().astype('Int64')

        resampled_non_adults['date_of_birth'] = pd.to_datetime({
            'year': resampled_non_adults['year_of_birth'],
            'month': resampled_non_adults['month_of_birth'],
            'day': resampled_non_adults['day_of_birth']
        }, errors='coerce')
        resampled_non_adults['date_of_vacc'] = pd.to_datetime({
            'year': resampled_non_adults['year_of_vacc'],
            'month': resampled_non_adults['month_of_vacc'],
            'day': resampled_non_adults['day_of_vacc']
        }, errors='coerce')

        resampled_non_adults['age_at_vaccination'] = (resampled_non_adults['date_of_vacc'] -
        resampled_non_adults['date_of_birth']).dt.days / 365.25

        # Replace non-adult rows with resampled valid rows
        synthetic_data.loc[non_adults.index, resampled_non_adults.columns] =
        resampled_non_adults

    # Finalize age column and remove temporary ones
    synthetic_data['age_on_first_vacc'] =
    synthetic_data['age_at_vaccination'].fillna(-1).round().astype(int)
    synthetic_data.loc[synthetic_data['age_on_first_vacc'] == -1, 'age_on_first_vacc'] =
    np.nan
    synthetic_data.dropna(subset=['age_on_first_vacc'], inplace=True)
    synthetic_data['age_on_first_vacc'] = synthetic_data['age_on_first_vacc'].astype(int)
    synthetic_data.drop(columns=['age_at_vaccination'], inplace=True)
    synthetic_data.drop(columns=['year_of_birth', 'month_of_birth', 'day_of_birth',
    'year_of_vacc', 'month_of_vacc', 'day_of_vacc'], inplace=True)
    synthetic_data.dropna(subset=['date_of_birth', 'date_of_vacc'], inplace=True)

    # ---------------------------------------------------------- #
    # STEP 6: Ensure first names are consistent with sex assignment #
    # ---------------------------------------------------------- #
    print("\n#########################################")
    print("\n STEP 6: Ensure sex matches gender-specific names and maintain statistical
    properties\n")
```

```
191
192 # Map numeric sex values to readable labels temporarily
193 synthetic_data['sex_source'] = synthetic_data['sex'].map({0: 'Female', 1: 'Male'})
194
195 # Replace first names with sex-consistent samples from source data
196 def assign_gender_consistent_name(row):
197     if row['sex_source'] == 'Male':
198         return np.random.choice(male_names_source)
199     elif row['sex_source'] == 'Female':
200         return np.random.choice(female_names_source)
201     else:
202         return np.nan
203
204 synthetic_data['first_name_consistent'] =
    synthetic_data.apply(assign_gender_consistent_name, axis=1)
205 synthetic_data['first_name'] = synthetic_data['first_name_consistent']
206 synthetic_data.drop(columns=['first_name_consistent', 'sex_source'], inplace=True)
207
208 # Show sex distribution comparison for validation
209 synthetic_sex_distribution = synthetic_data['sex'].value_counts(normalize=True)
210 print("\nSource Sex Distribution:")
211 print(sex_distribution_source)
212 print("\nInitial Synthetic Sex Distribution:")
213 print(synthetic_sex_distribution)
214
215 # ------------------------------------------------------- #
216 # STEP 7: Handle missing values in selected string fields #
217 # ------------------------------------------------------- #
218 print("\n##########################################")
219 print("\n STEP 7: Fill missing or blank last names, villages and occupation with random
    values from the source dataset\n")
220
221 # Fill blank/missing values with random samples from source
222 for col in ['last_name', 'village_name', 'occupation', 'vacc_manufacturer']:
223     random_values = source_data[col].dropna().unique()
224     blank_or_missing = (synthetic_data[col].isna()) | (synthetic_data[col] == '')
225     blank_count = blank_or_missing.sum()
226     if blank_count > 0:
227         synthetic_data.loc[blank_or_missing, col] = np.random.choice(random_values,
            size=blank_count)
228
229 # ------------------------------------------------------------- #
230 # STEP 8: Decode encoded values (categoricals) to source strings  #
231 # ------------------------------------------------------------- #
232 print("\n##########################################")
233 print("\n STEP 8: Re-encode numeric columns back to source string values\n")
234
235 # Use saved LabelEncoders to reverse-encode fields back to human-readable values
236 for col in ['first_name', 'last_name', 'village_name', 'occupation','vacc_manufacturer']:
237     le = source_value_maps[col]
238     value_mapping = {i: le.inverse_transform([i])[0] for i in range(len(le.classes_))}
239     synthetic_data[col] = synthetic_data[col].map(value_mapping).fillna('')
240
241 # Final decoding for sex and ID formatting
242 synthetic_data['sex'] = synthetic_data['sex'].replace({1: 'Male', 0: 'Female'})
243 synthetic_data['id'] = np.abs(synthetic_data['id'])  # Ensure positive values
244 synthetic_data['id'] = synthetic_data['id'].astype(int).astype(str)  # Convert to string
245
246 print("\nSynhetic data sample after re-encoding")
247 print(synthetic_data.head())
248
249 # ------------------------------------------------ #
250 # STEP 9: Save final synthetic dataset to a CSV    #
251 # ------------------------------------------------ #
252 print("\n##########################################")
253 print("\n STEP 9: Store the Synthetic Data to a CSV files\n")
254
255 # Map internal column names to output schema
256 columns_to_save = {
```

```
    'id': 'id',
    'first_name': 'first_name',
    'last_name': 'last_name',
    'sex': 'sex',
    'date_of_birth': 'dob',
    'village_name': 'village_name',
    'occupation': 'occupation',
    'date_of_vacc': 'covid_19_first_vacc_date',
    'age_on_first_vacc': 'age_on_first_vacc',
    'vacc_manufacturer': 'vacc_manufacturer'
}

# Select and rename columns for export
synthetic_data_for_csv =
synthetic_data[list(columns_to_save.keys())].rename(columns=columns_to_save)

# Preview final dataset and write to file
print("\nSynthetic data sample for CSV:")
print(synthetic_data_for_csv.head())
synthetic_data_for_csv.to_csv('synthetic_data_dswb_training.csv', index=False)
print("\n Synthetic data saved successfully to CSV file.")

########################################
# STEP 10: Store the Synthetic Data to a PostgreSQL Table
########################################
print("\n########################################")
print("\n STEP 10: Store the Synthetic Data to a PostgreSQL Table \n")
# Step 10a: Create the table if not exists
create_table_query = """
CREATE TABLE IF NOT EXISTS synthetic_dswb_training.synthetic_dataset_dswb_training
(
    id text,
    first_name text,
    last_name text,
    sex text,
    dob timestamp without time zone,
    village_name text,
    occupation text,
    covid_19_first_vacc_date timestamp without time zone,
    age_on_first_vacc integer,
    vacc_manufacturer text
);
"""

with engine.connect() as connection:
    connection.execute(text(create_table_query))  # Create table if it doesn't exist

# Step 10b: Prepare data for insertion with correct column names
synthetic_data_for_db = synthetic_data.rename(columns={
    'date_of_birth': 'dob',
    'date_of_vacc': 'covid_19_first_vacc_date'
})

# Step 10c: Insert synthetic data into the table
synthetic_data_for_db[['id', 'first_name', 'last_name', 'sex', 'dob', 'village_name',
'occupation', 'covid_19_first_vacc_date', 'age_on_first_vacc',
'vacc_manufacturer']].to_sql(
    'synthetic_dataset_dswb_training',
    con=engine,
    schema='synthetic_dswb_training',
    if_exists='append',  # Or 'replace' if you want to overwrite the table
    index=False
)
print("\n Synthetic data saved successfully to PostgreSQL table.")

print("################ End of Code ########################")
```