Author: Tan Thiam Huat, updated 5th August 2025

Documentation: Optimized Synthetic Data Generation Pipeline

This document describes the improved data processing flow for generating a privacy-compliant synthetic dataset using Microsoft Fabric, PySpark, and SDV (CTGAN). This optimized approach addresses high-cardinality issues and improves CTGAN performance while ensuring privacy compliance (GDPR, PCI-DSS).

Step 1: Import necessary libraries and Load Raw Data

Source: Files/transactions.csv stored in the Microsoft Fabric Lakehouse. (1 million rows)
Schema Example: CustomerName | Email | CardNumber | Age | Amount | Country | TransactionDate
Loads data into a Spark DataFrame with automatic schema inference.

```
from pyspark.sql.functions import col, sha2, concat_ws
import pandas as pd
from sdv.single_table import CTGANSynthesizer
from sdv.metadata import SingleTableMetadata

# ----------------------------
# LOAD RAW DATA
# ----------------------------
raw_df = spark.read.csv("Files/transactions.csv", header=True, inferSchema=True)
```

Step 2: Mask Direct Identifiers (Hashing)

Purpose: Remove PII by hashing CustomerName and Email with SHA-256 and dropping sensitive columns.

```
# ------------------------------------------------------------
# MASK DIRECT IDENTIFIERS (HASHING)
# ------------------------------------------------------------
masked_df = (
    raw_df
    .withColumn("CustomerHash", sha2(concat_ws(":", col("CustomerName"), col("Email")), 256))
    .drop("CustomerName", "Email", "CardNumber")   # Remove PII
)
```

Step 3: Sample Data to Pandas

Limit to 200,000 rows to improve CTGAN performance. Required since CTGAN works on pandas DataFrames.

```
# ------------------------------------------------------------
# SAMPLE DATA TO PANDAS
# ------------------------------------------------------------
sample_data = masked_df.limit(200_000).toPandas()
```

Step 4: Drop Unique Identifiers

CustomerHash is dropped to prevent extremely high cardinality which would cause CTGAN to create hundreds of thousands of one-hot encoded columns.

```
# ------------------------------------------------------------
# DROP UNIQUE IDENTIFIERS
# ------------------------------------------------------------
if "CustomerHash" in sample_data.columns:
    sample_data = sample_data.drop(columns=["CustomerHash"])
```

Step 5: Handle High-Cardinality Columns

- Group rare countries (<50 occurrences) into 'Other'.
- Convert TransactionDate to numeric features (DayOfWeek, Month) instead of treating every date as a unique category.

```
# ------------------------------------------------------------
# HANDLE HIGH-CARDINALITY COLUMNS
# ------------------------------------------------------------
# Reduce rare countries to 'Other'
if 'Country' in sample_data.columns:
    value_counts = sample_data['Country'].value_counts()
    rare_countries = value_counts[value_counts < 50].index
    sample_data['Country'] = sample_data['Country'].replace(rare_countries, 'Other')

# Extract date features
if 'TransactionDate' in sample_data.columns:
    sample_data['TransactionDate'] = pd.to_datetime(sample_data['TransactionDate'], errors='coerce')
    sample_data['DayOfWeek'] = sample_data['TransactionDate'].dt.dayofweek
    sample_data['Month'] = sample_data['TransactionDate'].dt.month
    sample_data = sample_data.drop(columns=['TransactionDate'])
```

Step 6: Detect and Update Metadata

Auto-detects data types, then explicitly sets:

- Age, Amount: numerical
- Country: categorical

```
# ------------------------------------------------------------
# DETECT AND UPDATE METADATA
# ------------------------------------------------------------
metadata = SingleTableMetadata()
metadata.detect_from_dataframe(data=sample_data)

# Ensure correct data types
if 'Age' in sample_data.columns:
    metadata.update_column('Age', sdtype='numerical')
if 'Amount' in sample_data.columns:
    metadata.update_column('Amount', sdtype='numerical')
if 'Country' in sample_data.columns:
    metadata.update_column('Country', sdtype='categorical')

print("✅ Metadata processed successfully.")
```

Step 7: Train CTGAN Synthesizer

Trains the CTGAN model for 80 epochs to learn statistical patterns of the dataset.

```
# ------------------------------------------------------------
# TRAIN CTGAN SYNTHESIZER
# ------------------------------------------------------------
synthesizer = CTGANSynthesizer(metadata, epochs=80, verbose=True)
synthesizer.fit(sample_data)
```

Step 8: Generate Synthetic Data

Creates 1,000,000 synthetic records and converts them back to Spark DataFrame for large-scale processing.

```
# ------------------------------------------------------------
# GENERATE SYNTHETIC DATA
# ------------------------------------------------------------
synthetic_data = synthesizer.sample(1_000_000)
synthetic_df = spark.createDataFrame(synthetic_data)

print(f"✅ Generated synthetic dataset with {synthetic_df.count():,} rows.")
```

Step 9: Apply Differential Privacy

Adds noise to Age and Amount columns to strengthen privacy guarantees using epsilon=0.5.

```
# ------------------------------------------------------------
# APPLY DIFFERENTIAL PRIVACY (NOISE INJECTION)
# ------------------------------------------------------------
epsilon = 0.5
dp_df = (
    synthetic_df
    .withColumn("Age", (col("Age") + (rand() * 2 - 1) / epsilon).cast("int"))
    .withColumn("Amount", spark_round(col("Amount") + (rand() * 10 - 5) / epsilon, 2))
)

print("✅ Differential privacy noise applied.")
```

Step 10: Save Dataset to Fabric Lakehouse

Final synthetic dataset is stored as a Delta table for analytics and ML pipelines.

```
# ------------------------------------------------------------
# SAVE PRIVACY-COMPLIANT SYNTHETIC DATASET TO FABRIC LAKEHOUSE
# ------------------------------------------------------------
dp_df.write.format("delta").mode("overwrite").saveAsTable('synthetic_transactions_delta')
```

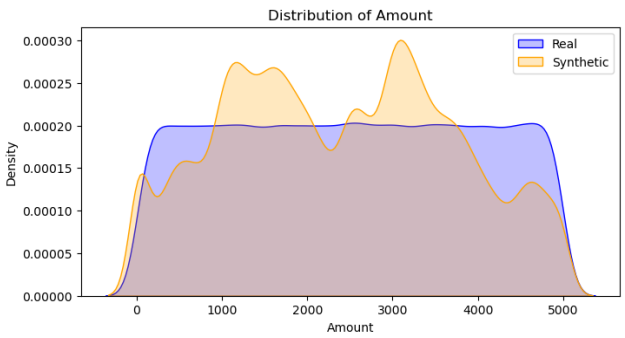
Flow Validation

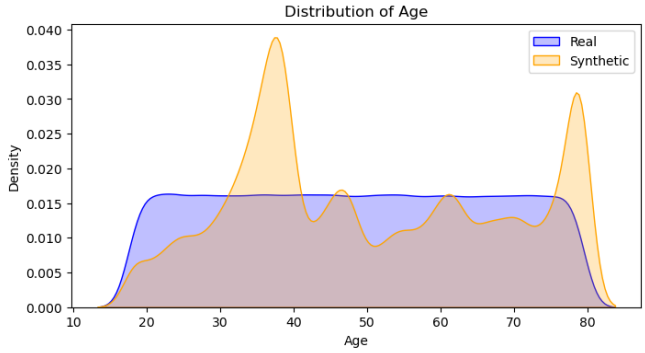
- PII is securely hashed and dropped before model training.
- Unique identifiers are removed to prevent dimensionality explosion.
- High-cardinality and date columns are optimized for CTGAN.
- Metadata is explicitly updated for better model quality.
- GAN-based synthetic data generation ensures realistic but non-identifiable data.
- Differential privacy noise adds extra protection to numeric fields.
- Data is stored in Delta Lakehouse, ready for analytics and ML pipelines.

Output

```
[9]
from sdv.evaluation.single_table import run_diagnostic
diagnostic = run_diagnostic(
    real_data=real_data,
    synthetic_data=synthetic_data,
    metadata=metadata
)
✓ - Command executed in 339 ms by TAN JIA HUI, JOY on 10:12:23 PM, 8/04/25

Generating report ...

(1/2) Evaluating Data Validity: |██████████| 5/5 [00:00<00:00, 77.84it/s]|
Data Validity Score: 100.0%

(2/2) Evaluating Data Structure: |██████████| 1/1 [00:00<00:00, 665.55it/s]|
Data Structure Score: 100.0%

Overall Score (Average): 100.0%
```

```
[6]
from sdv.evaluation.single_table import evaluate_quality
score = evaluate_quality(real_data, synthetic_data, metadata)
print(score)
✓ <1 sec - Command executed in 933 ms by TAN JIA HUI, JOY on 9:30:21 AM, 8/05/25

Generating report ...

(1/2) Evaluating Column Shapes: |██████████| 5/5 [00:00<00:00, 23.64it/s]|
Column Shapes Score: 89.17%

(2/2) Evaluating Column Pair Trends: |██████████| 10/10 [00:00<00:00, 34.47it/s]|
Column Pair Trends Score: 78.59%

Overall Score (Average): 83.88%
```





Recommended Enhancements

1. Automate rare category threshold selection based on dataset size.
2. Store metadata JSON for reproducibility.
3. Parameterize epsilon for configurable privacy levels.
4. Add quality evaluation metrics for synthetic data fidelity.
5. Integrate the entire pipeline as an automated Fabric notebook task.

6. The above output is **not ideal** because the synthetic data should be closer to the real distribution. However, it is common CTGAN to produce this pattern without tuning. Further research needs to be done, for example, increasing the **training epochs** (e.g., 150–200) for better convergence. Or **Data Normalization/Scaling** for Amount and Age before training (GANs work better with normalized inputs).