Generating Relational Data (Step by Step)

**1. Import Required Libraries**

1. import pandas as pd
2. from sdv.single\_table import CTGANSynthesizer
3. from sdv.metadata import SingleTableMetadata
4. from sdv.evaluation.single\_table import run\_diagnostic, evaluate\_quality, get\_column\_plot
5. from sdv.sampling import Condition

* **Pandas**: Used for data manipulation and analysis. It provides data structures like DataFrame that are essential for handling tabular data.
* **CTGANSynthesizer**: Part of the SDV (Synthetic Data Vault) library, this synthesizer uses a Conditional Generative Adversarial Network (GAN) to generate synthetic tabular data. GANs are a class of machine learning frameworks designed by Goodfellow et al., where two neural networks contest with each other in a game (the generative and the discriminative networks).
* **SingleTableMetadata**: This class is used to define and manage metadata for a single table. Metadata includes details about the column types, constraints, and other structural information of the dataset.
* **run\_diagnostic, evaluate\_quality, get\_column\_plot**: These functions are part of the SDV library's evaluation module. They are used to assess the quality and integrity of the generated synthetic data by comparing it with the real data.
* **run\_diagnostic**: Performs integrity checks and generates a diagnostic report highlighting potential issues or anomalies between the real and synthetic datasets.
* **evaluate\_quality**: Evaluates the overall quality of the synthetic data, providing various metrics.
* **get\_column\_plot**: Generates visual plots comparing the distributions of specific columns in the real and synthetic datasets.
* **Condition**: A class used to define conditions for conditional sampling. This allows for generating synthetic data that meets specific criteria based on values of certain columns.

**2. Load Real Data**

* Assume we have a CSV file customer\_data.csv with the following content:
  1. customer\_id,age,gender,purchase\_amount
  2. 1,35,M,100.50
  3. 2,28,F,75.20
  4. 3,42,F,150.75
  5. 4,30,M,200.30
  6. 5,50,M,300.25
  7. 6,45,F,180.60
  8. 7,32,F,90.80
  9. 8,55,M,250.25
  10. 9,38,M,120.40
  11. 10,29,F,80.50
* Load this data into a pandas DataFrame:
  1. data = pd.read\_csv('./data/customer\_data.csv')
  2. real\_data = pd.DataFrame(data)
* **Loading Data:** pd.read\_csv reads the CSV file located at ./data/customer\_data.csv and loads it into a pandas DataFrame. This step is crucial as it provides the real data that will be used for training the synthesizer and for evaluation purposes.

**3. Initialize the Metadata Object**

1. metadata = SingleTableMetadata()
2. metadata.detect\_from\_dataframe(data)

* **SingleTableMetadata**: This object is created to manage metadata for the data table.
* detect\_from\_dataframe: This method automatically detects and defines metadata from the provided DataFrame. It includes identifying column types (e.g., numerical, categorical), primary keys, and any constraints present in the data.

**4. Initialize the Synthesizer**

1. synthesizer = CTGANSynthesizer(metadata)

* **CTGANSynthesizer**: This synthesizer is initialized with the metadata object. It uses a Conditional GAN model to learn from the real data and generate synthetic data that adheres to the same structure and constraints specified in the metadata.

**5. Fit the Synthesizer to the Real Data**

1. synthesizer.fit(real\_data)

* **fit Method**: This method trains the CTGANSynthesizer on the real data. The model learns the distribution and relationships between different columns in the dataset. This process involves:
  + **Generator Network**: Tries to create synthetic data that mimics the real data.
  + **Discriminator Network**: Tries to distinguish between real and synthetic data. The two networks iteratively improve until the generator produces realistic synthetic data that the discriminator cannot easily distinguish from real data.

**6. Generate Synthetic Data**

1. synthetic\_data = synthesizer.sample(num\_rows=1000)

* **sample Method**: This method generates new synthetic data based on the learned patterns from the real data. In this case, 1000 rows of synthetic data are created. The generated data should maintain the statistical properties and relationships present in the real data.

**7. Perform Integrity Checks**

1. diagnostic\_report = run\_diagnostic(real\_data=real\_data, synthetic\_data=synthetic\_data, metadata=metadata)
2. print(diagnostic\_report)

* **run\_diagnostic**: This function compares the real and synthetic data, checking for integrity issues and potential anomalies. It generates a diagnostic report that provides insights into how well the synthetic data mimics the real data in terms of structure and consistency.

**8. Evaluate the Quality of the Synthetic Data**

1. quality\_report = evaluate\_quality(real\_data=real\_data, synthetic\_data=synthetic\_data, metadata=metadata)
2. print(quality\_report)

* **evaluate\_quality**: This function evaluates the quality of the synthetic data by comparing it to the real data. It produces a quality report that includes various metrics such as coverage, similarity, and fidelity, helping to assess the effectiveness of the synthetic data generation process.

**9. Generate Column Plot**

1. column\_name = 'purchase\_amount'
2. if column\_name in metadata.columns:
3. fig = get\_column\_plot(real\_data=real\_data, synthetic\_data=synthetic\_data, metadata=metadata, column\_name=column\_name)
4. fig.show()
5. else:
6. print(f"Column '{column\_name}' not found in the metadata.")

* **get\_column\_plot**: This function generates a plot comparing the distribution of a specified column (purchase\_amount in this case) in both the real and synthetic datasets. This visualization helps in understanding how well the synthetic data replicates the real data's distribution for that particular column.

**10. Define Conditions for Conditional Sampling**

1. condition = Condition(num\_rows=100, column\_values={'purchase\_amount': 250.25})

* **Condition Object**: Defines specific conditions for sampling. Here, it specifies that we want 100 rows of synthetic data where the purchase\_amount is approximately 250.25. Conditional sampling is useful for generating targeted synthetic samples based on specific criteria.

**11. Sample Data Based on Conditions**

1. conditional\_synthetic\_data = synthesizer.sample\_from\_conditions(conditions=[condition])
2. print(conditional\_synthetic\_data.head())

* **sample\_from\_conditions**: This method generates synthetic data that meets the specified conditions. In this example, it generates 100 rows of data where purchase\_amount is around 250.25. This feature is particularly useful for creating synthetic datasets for specific scenarios or use cases where certain conditions need to be met.

**Summary**

1. **Metadata Management**: Ensures that the structure and types of the data are correctly handled during synthetic data generation.
2. **Training and Sampling**: The CTGANSynthesizer learns from real data and generates synthetic data that retains the statistical properties and relationships of the original data.
3. **Evaluation**: Functions like run\_diagnostic, evaluate\_quality, and get\_column\_plot help assess the quality and fidelity of the synthetic data.
4. **Conditional Sampling**: Allows for generating synthetic data that meets specific criteria, enhancing the utility of the synthetic data in various applications.

For further details, you can refer to the [SDV Documentation](https://docs.sdv.dev/sdv/single-table-data/modeling/synthesizers/ctgansynthesizer/).

Resources for this lecture

relational\_data.py

customer\_data.csv