1. Generative AI

Generative AI uses ML models, including Generative Pre-trained Transformer (GPT), Generative Adversarial Networks (GANs), and Variational Auto-Encoders (VAEs). The models’ algorithms learn from existing data to create an entirely new [synthetic dataset](https://www.k2view.com/blog/synthetic-dataset/) that closely resembles the original.

* + GPT is a language model trained on extensive tabular data, capable of generating lifelike synthetic tabular data. GPT-based synthetic data generation solutions rely on understanding and replicating patterns from the training data, useful for augmenting tabular datasets and generating realistic tabular data for ML tasks.
  + GANs are based on "generator" and "discriminator" neural networks. While the generator creates realistic synthetic data, the discriminator distinguishes real data from synthetic data. During training, the generator competes with the discriminator to produce data that attempts to fool the model, eventually resulting in a high-quality synthetic dataset that closely resembles real data.
  + VAEs are based on an "encoder" and a "decoder". The encoder encapsulates the patterns and characteristics of actual data into a summary of sorts. The decoder attempts to turn that summary back into realistic data. In terms of tabular data, VAEs create fake rows of information that still follow the same patterns and characteristics as the real data.

1. Rules engine

A rules engine creates data via user-defined business policies. Intelligence can be added to the generated data by referencing the relationships between the data elements, to ensure the relational integrity of the generated data.

1. Entity cloning

Entity cloning extracts the data for a selected business entity (e.g., specific customer or loan) from all underlying sources, masking and cloning it on the fly. Since unique identifiers are created for each cloned entity, it’s ideal for quickly generating the massive amounts of data needed for load and performance testing.

1. Data masking

[Data masking](https://www.k2view.com/what-is-data-masking/) retains the statistical properties and characteristics of the original production data, while protecting sensitive or personal information. It replaces private data with pseudonyms or altered values, ensuring privacy while preserving utility.

Here’s a synopsis of the pros and cons for each synthetic data generation method:

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Pros | Cons | Key reason for use |
| Generative AI | Speed (time to data) | * Limited by the diversity and size of the real data * May not generate the data needed for maximum testing coverage * Needs access to production data * Requires specialized skills | * Real data is scarce or non-existent * Need for complex data distributions * Requirement for diverse synthetic datasets |
| Rules engine | Creates large quantities of data, without having to access production data | * Requires detailed knowledge of the data and the logic needed to create it * Labor-intensive and time-consuming | * No access to production data * New functionality testing * Negative testing * Well-defined data generation process |
| Entity cloning | Instantly generates large datasets for testing and ML training | * Lacks variation and diversity * Can’t generate new information or scenarios * Can pose a privacy risk if the cloned data is not properly masked | Performance and load testing |
| Data masking | * Ensures data privacy * Maintains the statistical properties and distribution of the original data | * Runs the risk of re-identification * Might distort data, affecting its quality and integrity | * Software testing * Loading compliant data into data lakes and data warehouses for analytical workloads |

07

Generating synthetic banking transaction data is valuable for testing, training machine learning models, and validating financial systems without exposing real, sensitive information. Here are several approaches, along with their explanations, limitations, and examples:

### 1. Rule-Based Generation

\*\*Explanation:\*\* This approach uses predefined rules to generate transaction data. The rules are based on typical transaction patterns, such as common transaction amounts, frequencies, and types (e.g., deposits, withdrawals, transfers).

\*\*Limitations:\*\*

- \*\*Limited Variability:\*\* The generated data may lack the diversity and unpredictability of real-world data.

- \*\*Scalability Issues:\*\* As complexity increases, managing and updating rules becomes difficult.

\*\*Example:\*\*

```python

import random

def generate\_transaction(account\_id):

transaction\_types = ['deposit', 'withdrawal', 'transfer']

amount = round(random.uniform(10, 5000), 2)

transaction\_type = random.choice(transaction\_types)

return {

'account\_id': account\_id,

'transaction\_type': transaction\_type,

'amount': amount,

'timestamp': '2024-06-14T12:34:56'

}

account\_id = 12345

transaction = generate\_transaction(account\_id)

print(transaction)

```

### 2. Statistical Modeling

\*\*Explanation:\*\* This method uses statistical models to generate data based on the distribution of real transaction data. Techniques like Gaussian distributions or Poisson processes can model transaction amounts and frequencies.

\*\*Limitations:\*\*

- \*\*Complexity:\*\* Requires a good understanding of statistical methods and their application to financial data.

- \*\*Data Quality:\*\* Generated data might not capture intricate dependencies and anomalies present in real data.

\*\*Example:\*\*

```python

import numpy as np

def generate\_statistical\_transaction(account\_id):

amount = round(np.random.normal(loc=200, scale=50), 2) # Normal distribution

transaction\_type = np.random.choice(['deposit', 'withdrawal', 'transfer'])

return {

'account\_id': account\_id,

'transaction\_type': transaction\_type,

'amount': amount,

'timestamp': '2024-06-14T12:34:56'

}

account\_id = 12345

transaction = generate\_statistical\_transaction(account\_id)

print(transaction)

```

### 3. Agent-Based Modeling

\*\*Explanation:\*\* Simulates the behavior of individual agents (customers) based on certain rules and probabilities. Each agent has its profile and behavior patterns, creating a more dynamic and realistic dataset.

\*\*Limitations:\*\*

- \*\*Complexity:\*\* Building and tuning agent models can be complex and time-consuming.

- \*\*Scalability:\*\* Managing interactions between a large number of agents can be computationally intensive.

\*\*Example:\*\*

```python

class CustomerAgent:

def \_\_init\_\_(self, account\_id):

self.account\_id = account\_id

self.balance = 1000

def perform\_transaction(self):

transaction\_type = random.choice(['deposit', 'withdrawal'])

amount = round(random.uniform(10, 500), 2)

if transaction\_type == 'withdrawal' and amount > self.balance:

amount = self.balance

self.balance += amount if transaction\_type == 'deposit' else -amount

return {

'account\_id': self.account\_id,

'transaction\_type': transaction\_type,

'amount': amount,

'timestamp': '2024-06-14T12:34:56'

}

agent = CustomerAgent(12345)

transaction = agent.perform\_transaction()

print(transaction)

```

### 4. Machine Learning Models

\*\*Explanation:\*\* Uses machine learning techniques like GANs (Generative Adversarial Networks) to generate synthetic data that mimics the properties of real transaction data. These models learn from real data and generate new data points.

\*\*Limitations:\*\*

- \*\*Data Requirement:\*\* Requires a substantial amount of real data to train the models.

- \*\*Computational Resources:\*\* Training sophisticated models like GANs requires significant computational power.

\*\*Example:\*\*

```python

# Pseudocode for GAN-based generation

class TransactionGAN:

def \_\_init\_\_(self, real\_data):

self.real\_data = real\_data

# Initialize GAN components (generator and discriminator)

def train(self):

# Train GAN on real data

def generate\_transaction(self):

# Use trained generator to create a new transaction

return generated\_transaction

gan = TransactionGAN(real\_transaction\_data)

gan.train()

new\_transaction = gan.generate\_transaction()

print(new\_transaction)

```

### 5. Hybrid Approaches

\*\*Explanation:\*\* Combines multiple methods to leverage their strengths and mitigate their weaknesses. For instance, statistical models might generate the overall structure, and machine learning models add realistic details.

\*\*Limitations:\*\*

- \*\*Implementation Complexity:\*\* Combining methods requires careful integration and tuning.

- \*\*Resource Intensive:\*\* May require substantial computational and data resources.

\*\*Example:\*\*

```python

# Pseudocode for hybrid approach

def generate\_hybrid\_transaction(account\_id):

basic\_transaction = generate\_statistical\_transaction(account\_id)

detailed\_transaction = enhance\_with\_gan(basic\_transaction)

return detailed\_transaction

account\_id = 12345

transaction = generate\_hybrid\_transaction(account\_id)

print(transaction)

```

Each of these approaches offers unique advantages and challenges. The choice of method depends on the specific requirements, available data, and computational resources.