Preprocessing

To evaluate the appropriate normalization method for each feature, we used describe() and obtained the following output:

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
count	32561.000000	3.256100e+04	32561.000000	32561.000000	32561.000000	32561.000000
mean	38.581647	1.897784e+05	10.080679	1077.648844	87.303830	40.437456
std	13.640433	1.055500e+05	2.572720	7385.292085	402.960219	12.347429
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.178270e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.783560e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.370510e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000

- **Education_num** – Ordinal values, we normalize using the formula to preserve the order and spacing between values. $z_{if} = \frac{r_{if}-1}{M_f-1}$
- **Age, fnlwgt, hours-per-week** – Quantitative values with relatively close mean and median, so we normalize using **Standard Scaling**.
- **Capital-gain, capital-loss** – Positive quantitative values with an extreme maximum relative to the distribution, therefore, we first apply a **log transformation** and then normalize using **Standard Scaling**.

For the categorical columns, we applied one-hot encoding:

```
# one-hot encoding for the categorical cols
categorical_columns = ['workclass', 'education', 'marital-status', 'occupation', 'relationship', 'race', 'sex', 'native-country']
normalized_df = pd.get_dummies(df, columns=categorical_columns, drop_first=True)
```

We chose this method because there is no inherent order among the different values in these columns, making label encoding unsuitable.

We performed an 80-20 train-test split for all three partitions using 3 different seeds. Below are the resulting splits in terms of quantity and label distribution for each partition:

```
Split num. 1:
Train set size: 26048, Test set size: 6513
Train label distribution: {0: 0.7591753685503686, 1: 0.24082463144963145}
Test label distribution: {0: 0.7592507293106096, 1: 0.24074927068939045}

Split num. 2:
Train set size: 26048, Test set size: 6513
Train label distribution: {0: 0.7591753685503686, 1: 0.24082463144963145}
Test label distribution: {0: 0.7592507293106096, 1: 0.24074927068939045}

Split num. 3:
Train set size: 26048, Test set size: 6513
Train label distribution: {0: 0.7591753685503686, 1: 0.24082463144963145}
Test label distribution: {0: 0.7592507293106096, 1: 0.24074927068939045}
```

GAN Architecture

The generator and discriminator are implemented as fully connected neural networks with the following specifications:

Generator:

- Input: Random noise (z_dim=100).
- Three hidden layers with ReLU activation, allowing the network to learn complex representations while maintaining computational efficiency.
- Output layer with a Tanh activation function to generate synthetic samples.

Discriminator:

- Input: Real or synthetic data (data_dim features).
- Two hidden layers with LeakyReLU activation (α=0.2) to prevent dead neurons and ensure information flow, especially for negative values.
- Dropout (0.3) applied to improve generalization by preventing overfitting.
- Output layer with a Sigmoid activation to predict real/fake classification.

The loss function chosen for training is Binary Cross-Entropy (BCE), the common choice for GANs as we saw in class. The generator is trained to maximize log(D(G(z))), ensuring it generates realistic synthetic samples, and performed better from BCE on the generator.

To ensure robustness, we trained the GAN model separately on each of the three 80-20 train-test splits, using different random seeds. The training was performed for 7,000 epochs with a batch size of 64.

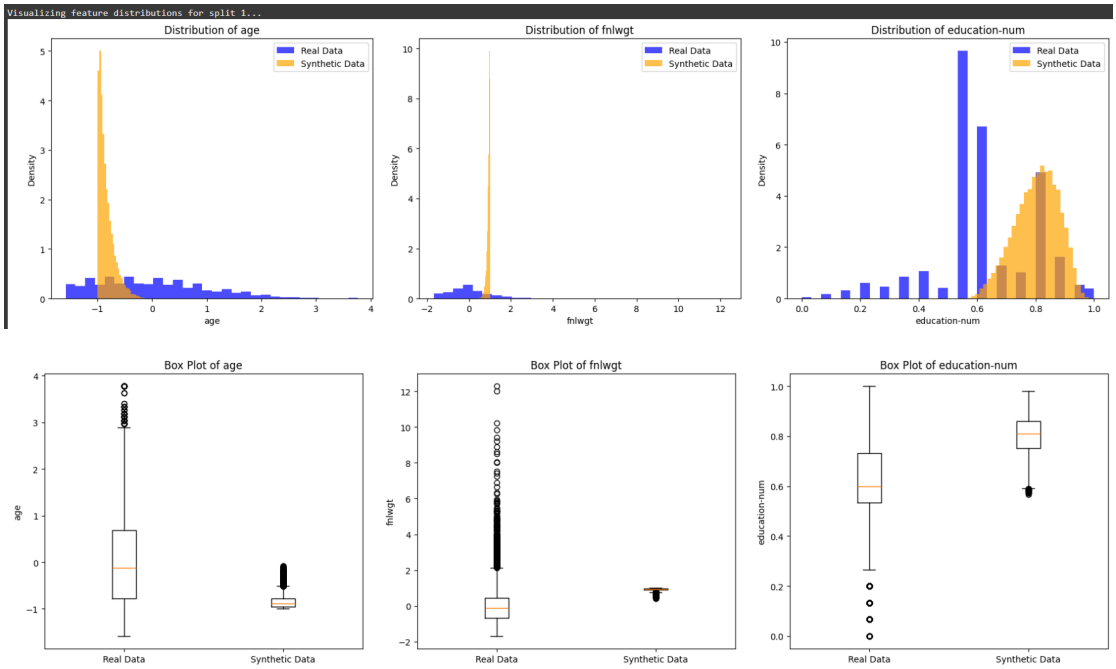
- **Discriminator Training:**
    - Receives both real and synthetic samples.
    - Computes losses separately for real and fake data.
    - Optimized using Adam (lr=0.0002), after experimenting with different learning rates.
- **Generator Training:**
    - Generates fake data from random noise.
    - Trained to maximize **log(D(G(z)))**, making it harder for the discriminator to distinguish real from fake.
    - Optimized using Adam (lr=0.0002).

We used Adam as the optimizer because it combines momentum and adaptive learning rates, making it well-suited for the unstable nature of GAN training. We

also experimented with different learning rates for the generator and discriminator but found that using the same lr=0.0002 for both provided the best results.

During training on the 3 training splits, we monitored both the discriminator and generator losses to assess convergence. Then we created synthetic samples.

Examples of the feature distributions of split #1 (see the full in the code):



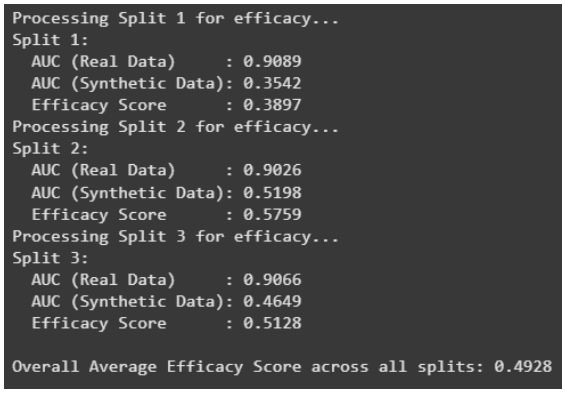
As observed, while the generated data captures some characteristics of the real data, there are noticeable discrepancies in certain feature distributions.

The **detection** metric evaluates whether the synthetic data is distinguishable from the real data. The Random Forest model, trained on a mixed dataset (50% real, 50% synthetic), achieved an AUC of 1 across all splits. This indicates that the classifier was able to perfectly differentiate between real and synthetic samples.

```
Processing Split 1...
Average AUC for Split 1: 1.0000
Processing Split 2...
Average AUC for Split 2: 1.0000
Processing Split 3...
Average AUC for Split 3: 1.0000

Overall Average AUC across all splits: 1.0000
```

The **efficacy** metric assesses whether synthetic data can serve as a substitute for real data in model training. The AUC when training on real data ranged from 0.9026 to 0.9089, while the AUC for models trained on synthetic data was significantly lower, ranging from 0.3542 to 0.5198. The overall average efficacy score was 0.4928, meaning that models trained on synthetic data retained less

than 50% of the predictive power of models trained on real data. This suggests that the synthetic data does not yet generalize well enough to be a reliable replacement for real data in predictive tasks.



Considering these findings, we decided to explore an alternative approach to improve the quality of synthetic data.

The "Forgiving Teacher" Approach

To improve the performance of the GAN model, we introduced a "Forgiving Teacher" approach, modifying the training dynamics of the discriminator. The key changes include:

1. Multiple Discriminators – Instead of a single discriminator, we trained three independent discriminators. Each discriminator learns different aspects of data distribution, making the feedback more diverse and reducing potential bias.
2. Generator Loss Averaging – Instead of relying on a single discriminator's feedback, the generator's loss is computed as the average of the logarithm of all three discriminators' outputs. This encourages the generator to produce synthetic samples that are realistic across multiple perspectives, making it more robust.
3. Independent Training for Each Discriminator – Each discriminator is optimized separately, preventing excessive domination by a single discriminator and ensuring a more balanced training process.

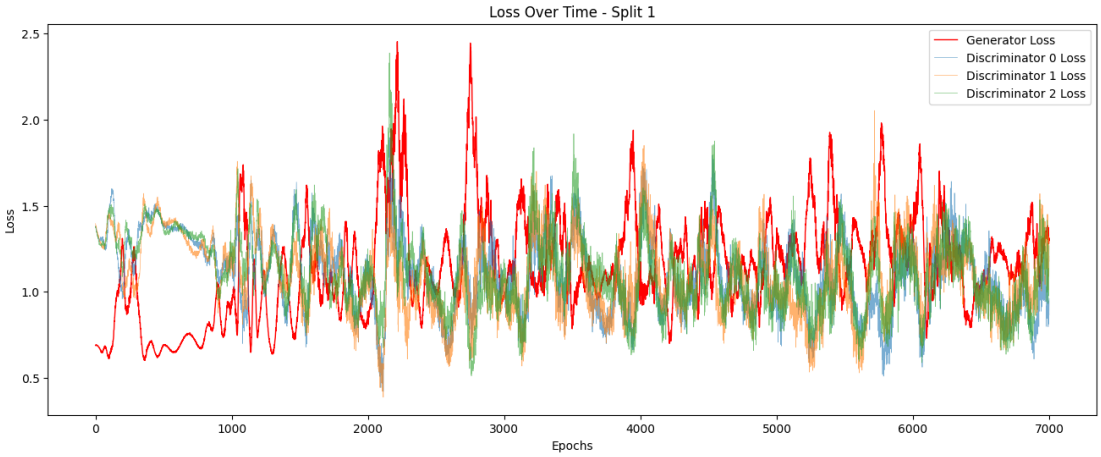
Using the Forgiving Teacher approach, we trained the GAN across three train-test splits for 7,000 epochs. The results show:

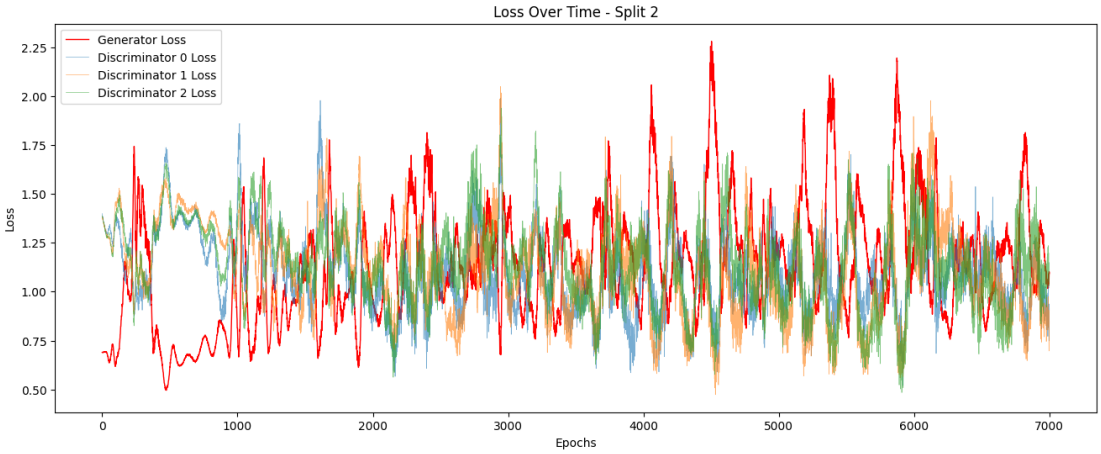
- Discriminator Losses: Initially high (~1.38 - 1.40), then gradually decreasing and stabilizing (0.9 - 1.3), indicating improved classification.
- Generator Loss: Starts low (~0.68 - 0.69) and increases over time (up to ~2.2), showing that the generator progressively produces more realistic samples.

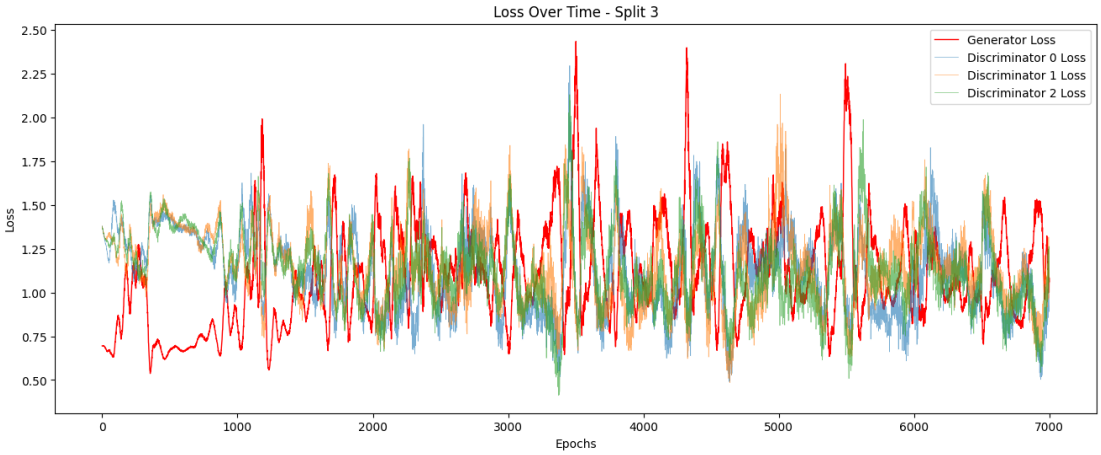
- Overall Stability: Losses remain balanced, avoiding collapse or overfitting. Multiple discriminators help with smooth training, preventing any single discriminator from dominating.

See the full training logs in Appendix No. 1.

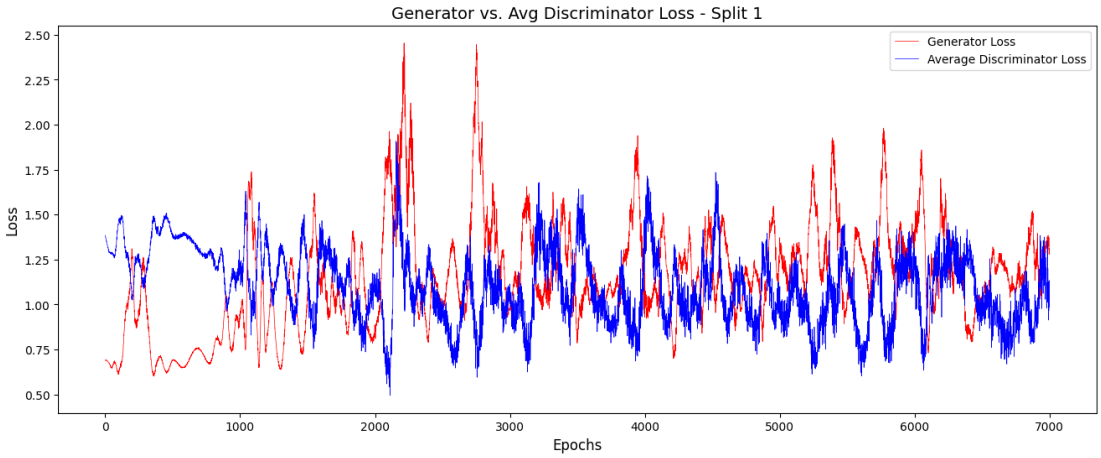
Training process graphs:

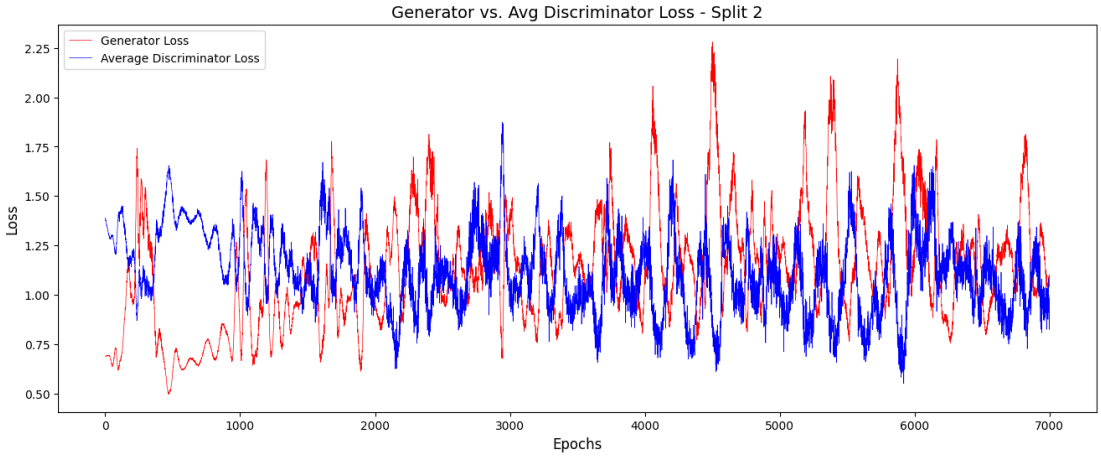


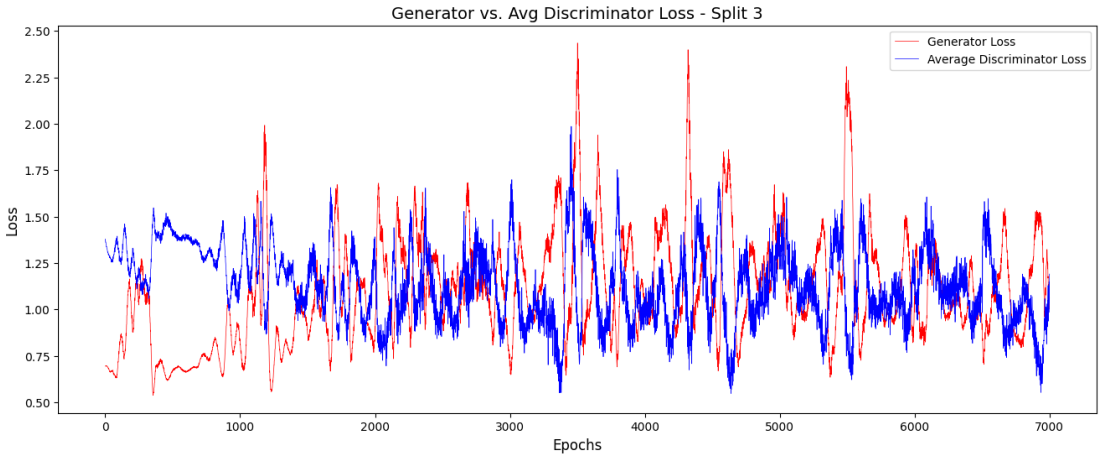




Training process graphs (with average discriminators loss):





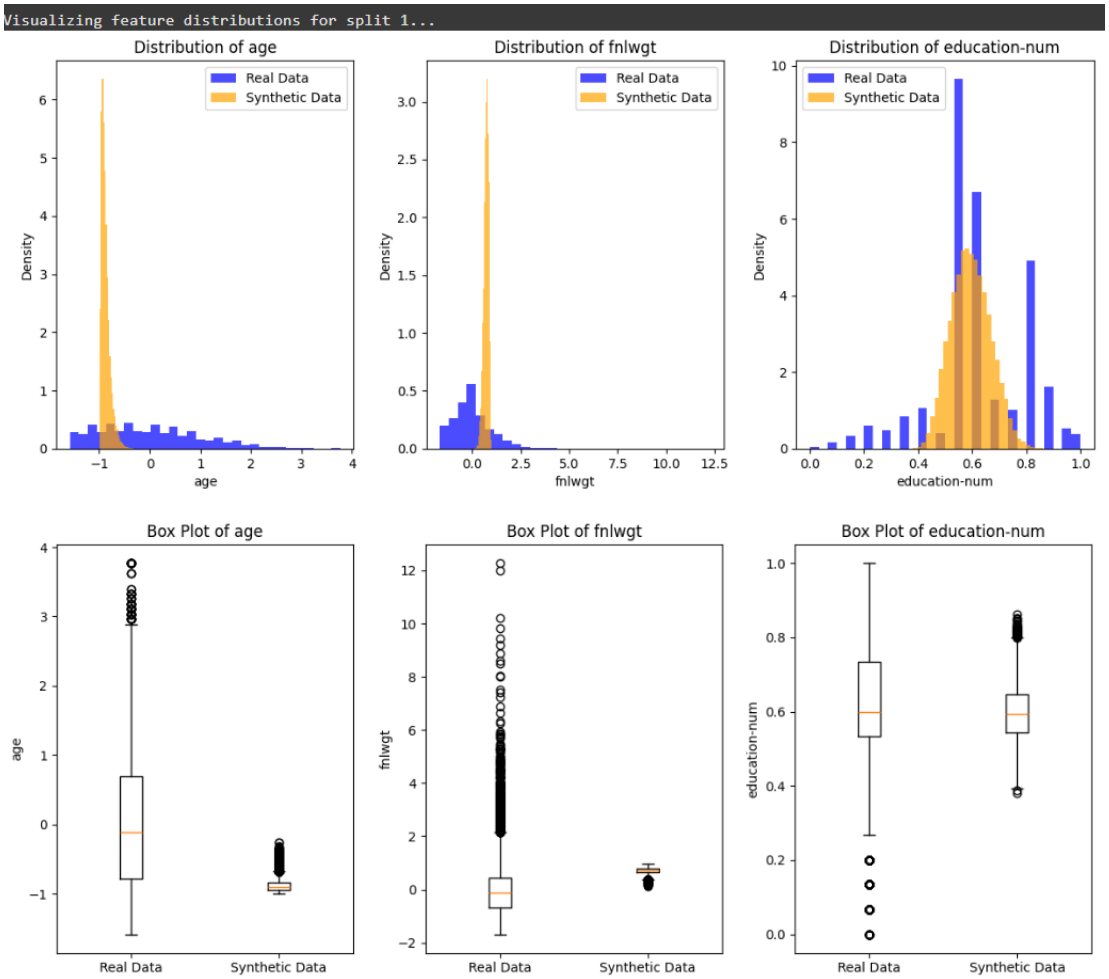


Afterward, following the guidelines given, we used the trained model to generate synthetic data.

The feature distribution visualizations indicate a notable improvement in the alignment between the synthetic and real data compared to the baseline GAN model. The density plots show that the generated data more accurately captures the underlying distribution of the real dataset. Additionally, the box plots

demonstrate a closer approximation of the statistical properties of the real data, though some discrepancies remain.

Examples of the feature distributions of split #1 (see the full in the code):



Detection

The detection results remained unchanged, with the Random Forest classifier still achieving a perfect AUC of 1.000 across all splits. This suggests that despite improvements in distribution alignment, the synthetic data still contains patterns that make it easily distinguishable from real data.

```
Processing Split 1...
Average AUC for Split 1: 1.0000
Processing Split 2...
Average AUC for Split 2: 1.0000
Processing Split 3...
Average AUC for Split 3: 1.0000

Overall Average AUC across all splits: 1.0000
```

Efficacy

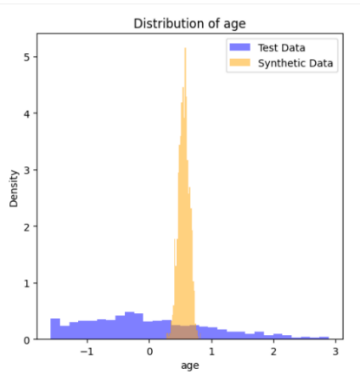
The efficacy score has significantly improved, reaching an average of 0.6345 compared to the previous 0.4928. This indicates that models trained on

synthetic data now retain a much larger portion of the predictive power of models trained on real data. Notably, splits 1 and 2 achieved efficacy scores above 0.72, demonstrating substantial progress in generating useful synthetic data for downstream tasks. This improvement suggests that the Forgiving Teacher approach effectively enhanced the quality of synthetic data.

```
Processing Split 1 for efficacy...
Split 1:
  AUC (Real Data)     : 0.9089
  AUC (Synthetic Data): 0.6867
  Efficacy Score      : 0.7556
Processing Split 2 for efficacy...
Split 2:
  AUC (Real Data)     : 0.9026
  AUC (Synthetic Data): 0.6546
  Efficacy Score      : 0.7252
Processing Split 3 for efficacy...
Split 3:
  AUC (Real Data)     : 0.9066
  AUC (Synthetic Data): 0.3833
  Efficacy Score      : 0.4228

Overall Average Efficacy Score across all splits: 0.6345
```

cGAN implementation

We decided to implement a cGAN to address challenges such as mode collapse, a common issue in traditional GANs. Mode collapse occurs when the generator learns to produce only a limited variety of outputs, failing to capture the full diversity of the data distribution.



Signs of Model Collapse **-** Gan output of feature age represents model collapse because the synthetic data distribution is tightly concentrated around a narrow range, while the real data distribution is spread across a wider range of values. This lack of diversity in the synthetic data indicates that the generator is repeatedly producing similar outputs instead of capturing the full variation of the real data.

How to solve it ? By conditioning both the generator and discriminator on an additional input, typically a label or instruction specifying the desired type of output, cGANs encourage the generator to produce outputs that correspond to each condition. This targeted generation helps prevent the network from

converging on repetitive samples by forcing it to learn distinct patterns for each label. Additionally, cGANs improve control over output, enabling the generation of data tailored to specific requirements, which is particularly useful for complex datasets with multiple categories. By implementing cGAN, we aim to improve both the stability and diversity of the generated data, mitigating the risk of model collapse while enhancing model performance and usability.

Architecture of cGAN

Generator Architecture

Inputs:

- A random noise vector z (e.g., of size 100).
- A condition vector (e.g., one-hot encoded label).

Processing Steps:

1. Concatenate the noise vector and condition vector along the feature axis.
2. Pass the concatenated input through multiple fully connected layers.
3. Use activation functions between layers to introduce non-linearity.
4. apply normalization technique to improve stability and convergence.
5. The final output layer applies a Tanh activation function to generate synthetic samples within a specified range.

Layer	Input Size	Output Size	Activation Function
Input Concatenation	(batch_size, data_dim) + (batch_size, label_dim)	(batch_size, data_dim + label_dim)	None
Layer 1	data_dim + label_dim	hidden_dim (e.g., 128)	Leaky ReLU (slope = 0.2)
Dropout	hidden_dim	hidden_dim	None (Dropout with p=0.3)
Layer 2	hidden_dim	hidden_dim	Leaky ReLU (slope = 0.2)
Dropout	hidden_dim	hidden_dim	None (Dropout with p=0.3)
Output Layer	hidden_dim	1	Sigmoid

Discriminator Architecture

Inputs:

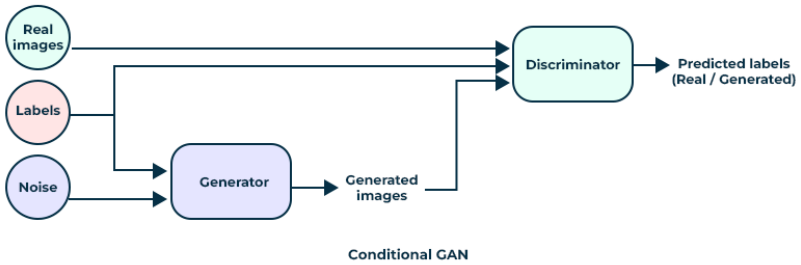
A data sample (either real or synthetic).

A condition vector (same type used for the generator).

Processing Steps:

1. Concatenate the data sample and condition vector along the feature axis.
2. Pass the concatenated input through several fully connected layers.
3. Apply Leaky ReLU activation functions to allow for small negative gradients and prevent vanishing gradients.
4. Use dropout layers to reduce overfitting and enhance model robustness.
5. The final output layer applies a Sigmoid activation function to produce a probability indicating whether the input is real or fake.

Layer	Input Size	Output Size	Activation Function
Input Concatenation	(batch_size, data_dim) + (batch_size, label_dim)	(batch_size, data_dim + label_dim)	None
Layer 1	data_dim + label_dim	hidden_dim (e.g., 128)	Leaky ReLU (slope = 0.2)
Dropout 1	hidden_dim	hidden_dim	None (Dropout with p=0.3)
Layer 2	hidden_dim	hidden_dim	Leaky ReLU (slope = 0.2)
Dropout 2	hidden_dim	hidden_dim	None (Dropout with p=0.3)
Output Layer	hidden_dim	1	Sigmoid



Training Parameters

Number of epochs: 7000

Batch size: 64

Noise dimension (z_dim): 100

Hidden dimension (hidden_dim): 128

Optimizer : Adam

Learning rate: 0.0002

Loss function : Binary Cross-Entropy Loss (BCE)

Results

Efficacy

The goal of efficacy is to determine whether your synthetic data is a useful substitute for the real data

```
Processing Split 1 for efficacy...
Split 1:
  AUC (Real Data)     : 0.9089
  AUC (Synthetic Data): 0.5450
  Efficacy Score      : 0.5997
Processing Split 2 for efficacy...
Split 2:
  AUC (Real Data)     : 0.9026
  AUC (Synthetic Data): 0.6373
  Efficacy Score      : 0.7061
Processing Split 3 for efficacy...
Split 3:
  AUC (Real Data)     : 0.9066
  AUC (Synthetic Data): 0.3362
  Efficacy Score      : 0.3708

Overall Average Efficacy Score across all splits: 0.5589
```

The efficacy scores across the splits show variability, with scores ranging from 0.3708 to 0.7061. While the real data consistently achieves high AUC values around 0.90, the synthetic data performs significantly worse, indicating that it does not capture the underlying patterns effectively. The overall average efficacy score of 0.5589 suggests that the synthetic data is less reliable for predictive modeling compared to real data.

Potential Reasons:

1. Incomplete Feature Relationships - The synthetic data may not fully capture important correlations and dependencies between features, leading to lower performance in predictive tasks.
2. Imperfect Label Conditioning - Since the cGAN is conditioned on the income labels, it may have failed to accurately model complex relationships between other features and the target variable, resulting in reduced efficacy.

Detection

The goal of the detection metric is to see whether your synthetic data is similar to the real one.

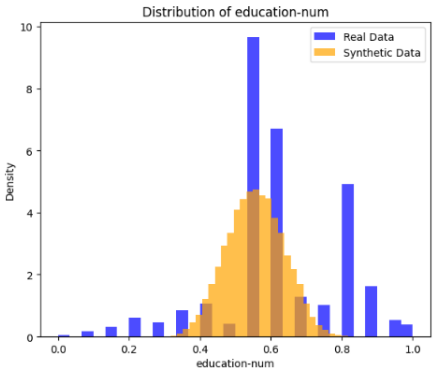
```
Processing Split 1...
Average AUC for Split 1: 1.0000
Processing Split 2...
Average AUC for Split 2: 1.0000
Processing Split 3...
Average AUC for Split 3: 1.0000

Overall Average AUC across all splits: 1.0000
```

The detection metrics indicate that the classifier perfectly distinguishes between real and synthetic data, with an AUC of 1.0000 for all splits. This suggests that the synthetic data generated by the cGAN does not resemble the real data in terms of feature distributions or relationships. The perfect separation highlights a limitation in the generator's ability to create realistic synthetic samples.

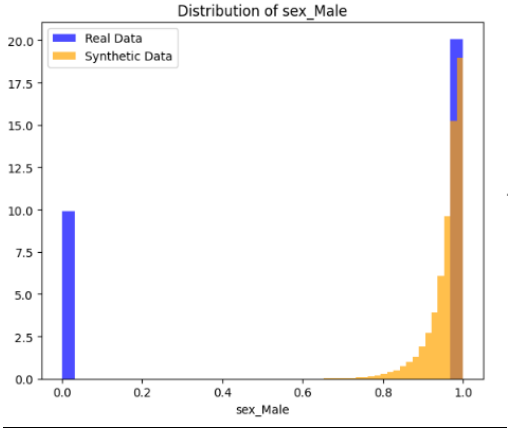
- **Weak Conditioning on Labels**- Since the cGAN was conditioned on the income labels, it may not have fully captured the relationships between other features, resulting in synthetic data with weak internal consistency.
- **Model or Training Limitations**- Insufficient training iterations, suboptimal hyperparameters, or inadequate model capacity may have hindered the generator's ability to produce high-quality synthetic data.

Addressing these issues through better model design, stronger label conditioning, and improved training strategies could help reduce the classifier's ability to distinguish between real and synthetic data.

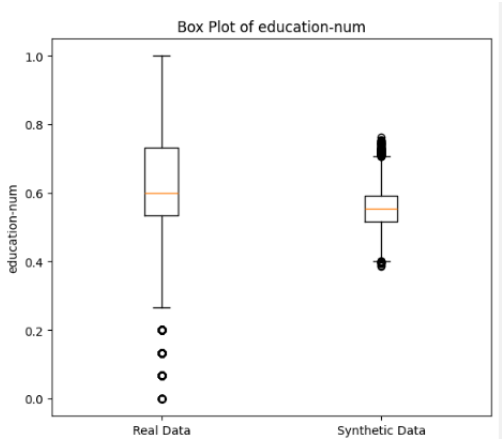
Compare between the synthetic to original by cGAN :



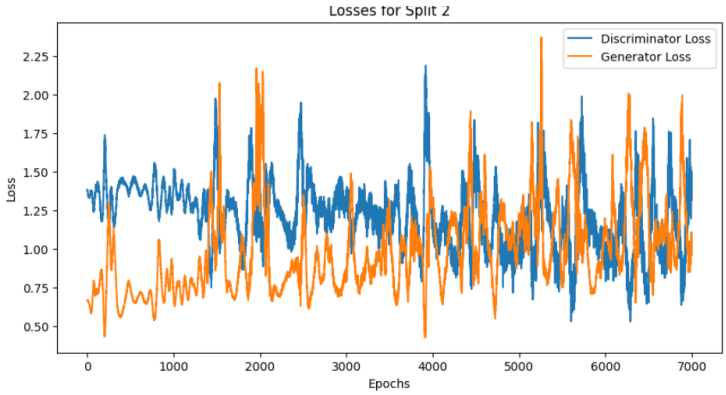
The graph shows that the real data has distinct peaks and gaps, indicating discrete education levels, while the synthetic data follows a smoother, more continuous distribution. Although the synthetic data captures the overall central tendency, it fails to replicate the key spikes and dips, highlighting the generator's difficulty in accurately modeling structured or categorical features. This suggests a need for improved handling of discrete data during training.



The real data for the sex_Male feature is binary, with clear values at 0 and 1, while the synthetic data shows a continuous distribution skewed toward 1. Since the cGAN was conditioned on the income labels and not directly on the sex_Male feature, it likely failed to properly capture the binary nature of this feature. This indicates that the generator did not effectively learn feature-specific distributions, potentially due to inadequate conditioning on related variables during training.



The box plot shows that the synthetic data for education-num has a narrower range and fewer extreme values compared to the real data. While the median values are somewhat similar, the synthetic data lacks the variability and outliers present in the real data. This raises questions about whether the cGAN effectively captured the full distribution of the feature or if mode collapse occurred, resulting in limited diversity in the synthetic samples. Improving the generator's ability to model feature variability may enhance synthetic data quality.



The loss graph for Split 2 shows significant fluctuations in both the discriminator and generator losses throughout training, indicating instability in the adversarial learning process. This instability may be caused by an imbalance in training dynamics, where either the generator or discriminator becomes too strong or too weak, leading to oscillations rather than convergence.

Possible Causes:

1. Learning Rate Imbalance: The learning rates for the generator and discriminator might not be optimized, causing one to overpower the other.
2. Model Imbalance: The network architecture may be uneven in capacity, leading to one model dominating the training process.

Suggested Solutions:

- Adjust Learning Rates: Experiment with different learning rates for both the generator and discriminator.
- Use Gradient Clipping: Limit large updates to weights to prevent drastic oscillations.