Autoencoders for Synthetic Tabular Data Generation

Generating synthetic augmented tabular data using autoencoders.

Dr. Wan Bae, Sartaj Bhuvaji

Introduction

Autoencoders are neural networks that encode input sample data into a smaller representation and re-generate similar data from this representation. This functionality makes them an ideal candidate for synthetic data generation.

Auto Encoders consist of two parts:
Encoder:
The encoder takes an input sample and maps it to a lower-dimensional representation known as the "latent space" or "bottleneck." The purpose of the encoder is to extract the most important features or patterns from the input data and encode them in a compressed form onto the latent space.

Decoder:
The decoder receives this latent representation and attempts to reconstruct the original input data. The decoder's objective is to generate an output that closely resembles the input sample.

Throughout the training process, the autoencoder fine-tunes its parameters to minimize the reconstruction error, ensuring that the decoded output closely aligns with the original input. This training mechanism facilitates the autoencoder in learning to encode the data into the latent space and then decode it back into its original form.

The autoencoder's remarkable ability to reconstruct data samples makes it an ideal candidate for synthetic data generation. As the decoder reconstructs values from the latent space, the generated results conform to the same distribution as the input data. This inherent capability allows autoencoders to effectively synthesize new data samples that share the characteristics and patterns of the original dataset.

Our Data

Our dataset comprised 26 patient data consisting of 27 min-max normalized [0–1] variable columns along with a binary label column labeled as ‘class’ [0,1]. Our dataset exhibited an imbalanced class distribution. To address this issue, we employed a data splitting strategy based on the ‘class’ label, resulting in two separate data frames: minority_df and majority_df. The minority_df contained the samples from the class with fewer instances, which naturally had a relatively smaller number of samples.

Given the imbalanced nature of the minority class, we recognized the importance of mitigating the class imbalance problem effectively. To achieve this, we employed the synthetic data generation approach using an autoencoder. Our focus was to train the autoencoder exclusively on the minority_df, as this approach allowed the model to learn the underlying patterns specific to the minority class.

Architecture

Single Encoder

The single-encoder approach focused on exploring different layer structures with a single-encoder layer containing eight to twenty nodes. This meant that the encoder portion of the autoencoder consists of a single layer responsible for mapping the input data to a lower-dimensional representation. A higher number of nodes allows for a more detailed encoding of the input data, capturing finer patterns and nuances, but might lead to overfitting.

By experimenting with different layer structures, we can investigate how the number of nodes in the single encoder layer affects the autoencoder’s performance. It’s important to note, that the choice of the number of nodes depends on the specific dataset and the complexity of the underlying patterns in the data.

For our dataset, we experimented with different combinations of below:

```
encoder_dense_layers_trial = [[8], [10], [12], [14], [16], [18] ,[20]]

bottle_neck_trial = [8, 10, 12, 14, 16, 18]

decoder_dense_layers_trial = [[8, 10], [10, 12], [12, 14], [14, 16], [16, 18], [18, 20],
[20,22]]
```

Balanced

In the balanced approach, our autoencoder model consists of two encoder layers and two decoder layers. This meant that the encoding and decoding processes are performed in multiple steps, allowing for a more intricate representation of the input data. The two encoder layers progressively map the input data to a lower-dimensional latent space. Each encoder layer extracts and encodes different levels of abstraction and features from the input data. The first encoder layer captures more high-level features, while the second encoder layer further refines the representation by capturing more fine-grained details.

After the data is encoded in the latent space, the two decoder layers work in reverse, gradually reconstructing the original input data from the encoded representation. Similar to the encoder layers, each decoder layer contributes to the reconstruction process at different levels of abstraction. The balanced approach with two encoder and two decoder layers provides a more expressive and flexible architecture for the autoencoder. It allows for capturing both high-level and low-level details in the input data, leading to more accurate reconstructions.

The specific configuration of the encoder and decoder layers, including the number of layers, will depend on the complexity of the dataset and the desired level of representation. Experimenting with different layer sizes and structures can help determine the optimal architecture for the balanced autoencoder model.

For our dataset, we experimented with different combinations of below:

```
encoder_dense_layers_trial = [[10, 8], [12, 10], [14, 12], [16, 14], [18, 16], [20, 18] ,[22,20]]

bottle_neck_trial = [8, 10, 12, 14, 16, 18]

decoder_dense_layers_trial = [[8, 10], [10, 12], [12, 14], [14, 16], [16, 18], [18, 20], [20,22]]
```

Heavy Decoder

In the heavy decoder approach, the focus is on enhancing the quality of synthetic data generation by placing more emphasis on the decoder part of the autoencoder. In our approach, the autoencoder model is designed with two encoder layers and four decoder layers. The two encoder layers are responsible for mapping the input data to a lower-dimensional latent space, as in the previous approaches.

On the other hand, the four decoder layers in the heavy decoder approach are tasked with reconstructing the output as close as to the original input data from the latent space representation. By increasing the number of decoder layers, the model can capture more intricate details and fine-grained patterns during the data generation process. This allows for a more comprehensive and nuanced synthesis of the synthetic data.

The heavier decoder architecture provides the autoencoder with more flexibility and expressive power to generate high-quality synthetic data that closely resembles the original dataset. The additional layers in the decoder allow for a richer reconstruction process, enabling the model to capture complex relationships and dependencies present in the input data. But as the depth of the decoder increases, it becomes susceptible to overfitting.

During the training phase, the autoencoder optimizes its parameters to minimize the reconstruction error between the input data and the output generated by the decoder. By adjusting the weights and biases in both the encoder and decoder layers, the model learns to generate synthetic data that exhibit similar characteristics to the original dataset.

It's important to note that the specific configuration of the layers, including the number of nodes in each layer, should be determined through experimentation and tuning. The optimal architecture depends on the complexity of the dataset and the desired quality of the generated synthetic data.

For our dataset, we experimented with different combinations of below:

encoder_dense_layers_trial = [[10, 8], [12, 10], [14, 12], [16, 14], [18, 16], [20, 18],[22, 20]]

bottle_neck_trial = [8, 10, 12, 14, 16, 18]

decoder_dense_layers_trial = [[6, 8, 10, 12], [8, 10, 12, 14], [10, 12, 14, 16], [12, 14, 16, 18], [14, 16, 18, 20], [16, 18, 20, 22], [18, 20, 22, 24]]

Loss Function

The loss function of an autoencoder measures the discrepancy between the input data (Y) and the output generated by the decoder (Ŷ). The goal of the autoencoder is to minimize this loss function during the training process, which leads to the model learning to reconstruct the input data as accurately as possible.

The most commonly used loss function for autoencoders is the mean squared error (MSE). The MSE loss calculates the average squared difference between the original input data and the output generated by the decoder.

$$\mathrm{MSE} = \frac{1}{n} \sum_{i=1}^{n} \left(Y_i - \hat{Y}_i \right)^2 .$$

Kullback-Leibler Divergence

In our study, we employed the Kullback-Leibler (KL) divergence to assess the similarity between the distribution of real data and synthetically generated data.

The KL divergence, denoted as KL(P || Q), is a mathematical measure that quantifies the difference between two probability distributions, P and Q. Specifically, it calculates the information lost when using distribution Q to approximate distribution P.

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)} \right)$$

Where:

P(x) is the probability of observing data point x in the original data distribution.
Q(x) is the probability of observing data point x in the synthetic data distribution.
The summation is taken over all possible data points x in the dataset.

A crucial property of KL divergence is that it is non-negative; hence,
KL(P || Q) ≥ 0. If the two distributions are identical, the KL divergence will be zero, indicating a perfect match between the real and synthetic data. Conversely, a higher KL divergence value suggests a greater dissimilarity between the distributions.

Similarity

A row (R) is said to be similar to row (R`) if all the values of all the respective columns in the row (R`) are within the *threshold* % deviation of all the values of all the same respective columns of row (R).

When generating a synthetic dataset, it becomes crucial to assess the presence of duplicate and similar rows to ensure the quality and usability of the generated data. High levels of duplication often indicate that the model is overfitting, meaning it has memorized the training data rather than capturing meaningful patterns for generalization.

A well-performing synthetic data generation model should exhibit a low number of duplicate rows while maintaining a sufficient level of similarity to the original data. Striking this balance ensures that the generated dataset accurately captures the underlying patterns and characteristics of the real data, enabling reliable and insightful analysis.

For our experiments, we calculated similarity with thresholds of 10%, 25%, and 50% to identify similar rows generated by the synthetic data.

Training Parameters

Optimizer = Adam (learning_rate = 0.001)
Epochs = 150
Batch size = 16
Validation split = 0.25

Results

Single Encoder Approach

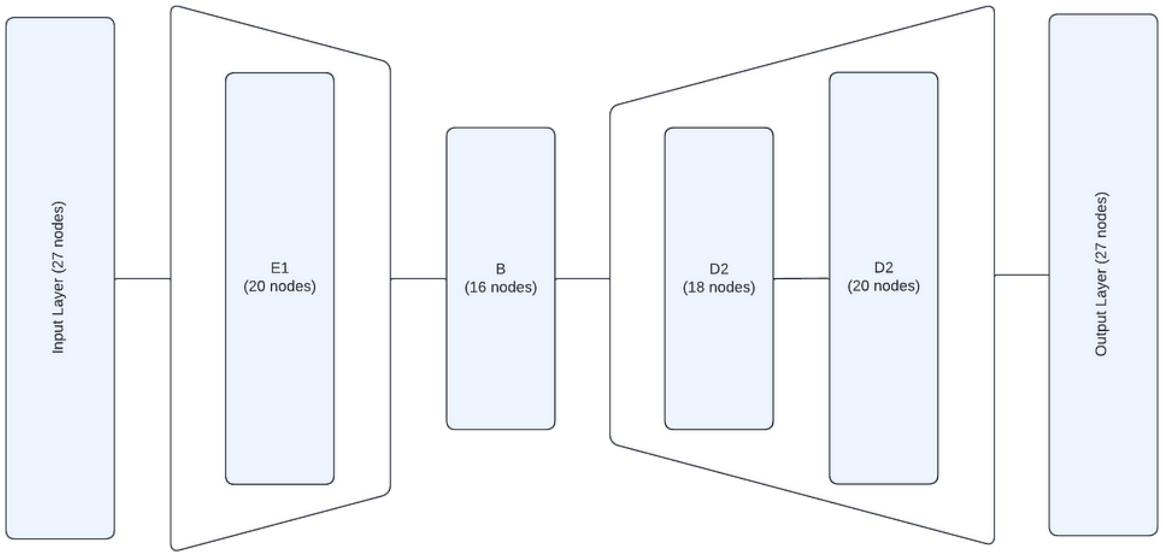
The evaluation of different models on our dataset indicates that the single-encoder model outperformed the heavy decoder and the balanced approach. This finding is particularly relevant considering that our dataset was small and could be adequately represented by two decoder layers.

The single encoder model's superior performance suggests that the complexity of the heavy decoder architecture might not have been necessary for our dataset. It is possible that the heavy decoder, with its increased number of layers, might have been too large or overly complex for capturing the specific patterns and characteristics present in our data.

> In cases, where data is too limited, a simpler model, like the single encoder with two decoder layers, can be more effective. The single encoder model likely struck a better balance between model complexity and data representation for the dataset, resulting in better performance.

It's worth noting that the performance of different models can vary depending on the dataset and the specific characteristics of the data. The optimal model architecture and layer configurations may differ across different datasets and applications.

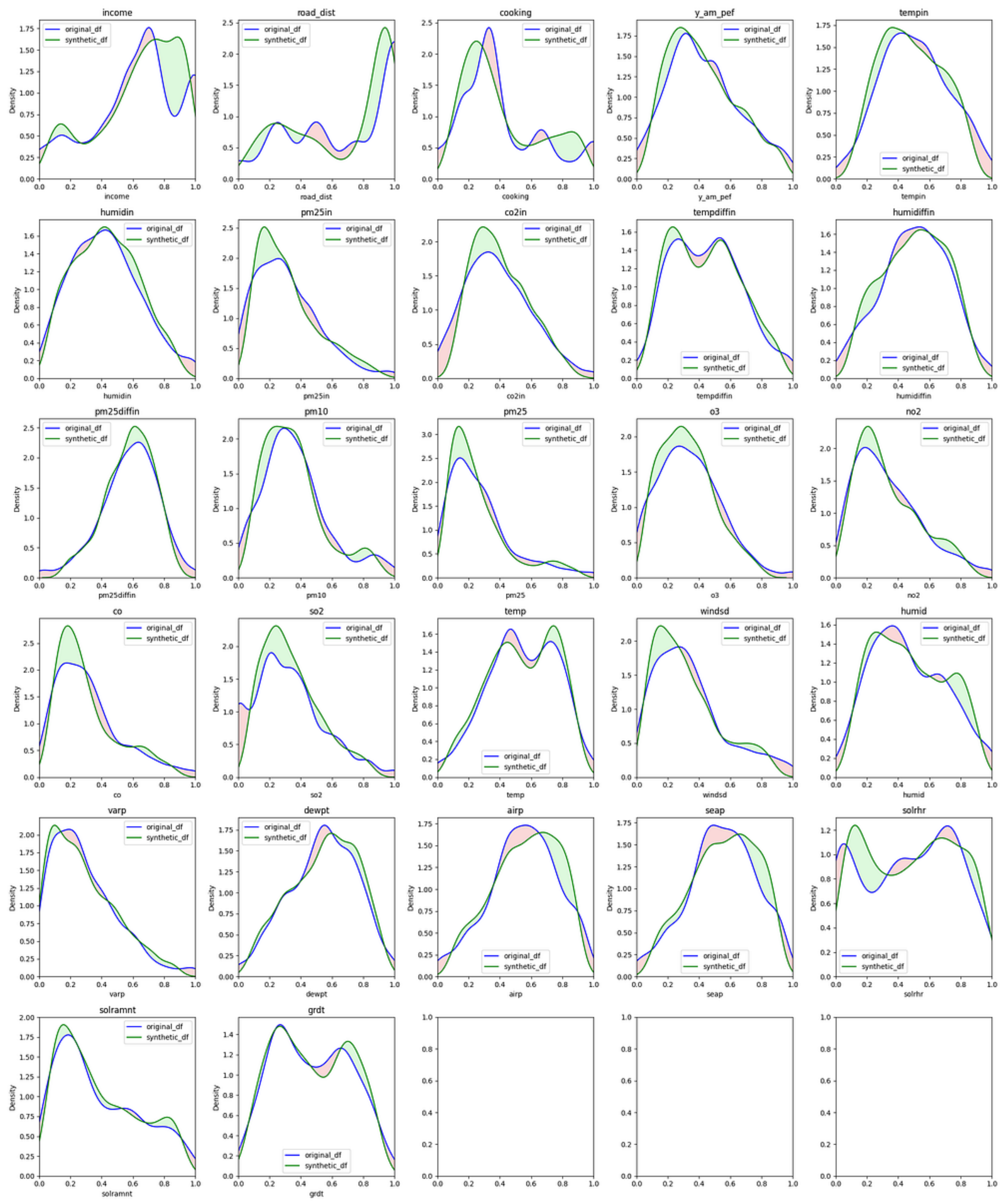
Based on our evaluation, the single encoder model with two decoder layers is the most suitable choice for our dataset. The single-encoder models with different architectures outperformed the deep-structured heavy decoder and balanced approaches.



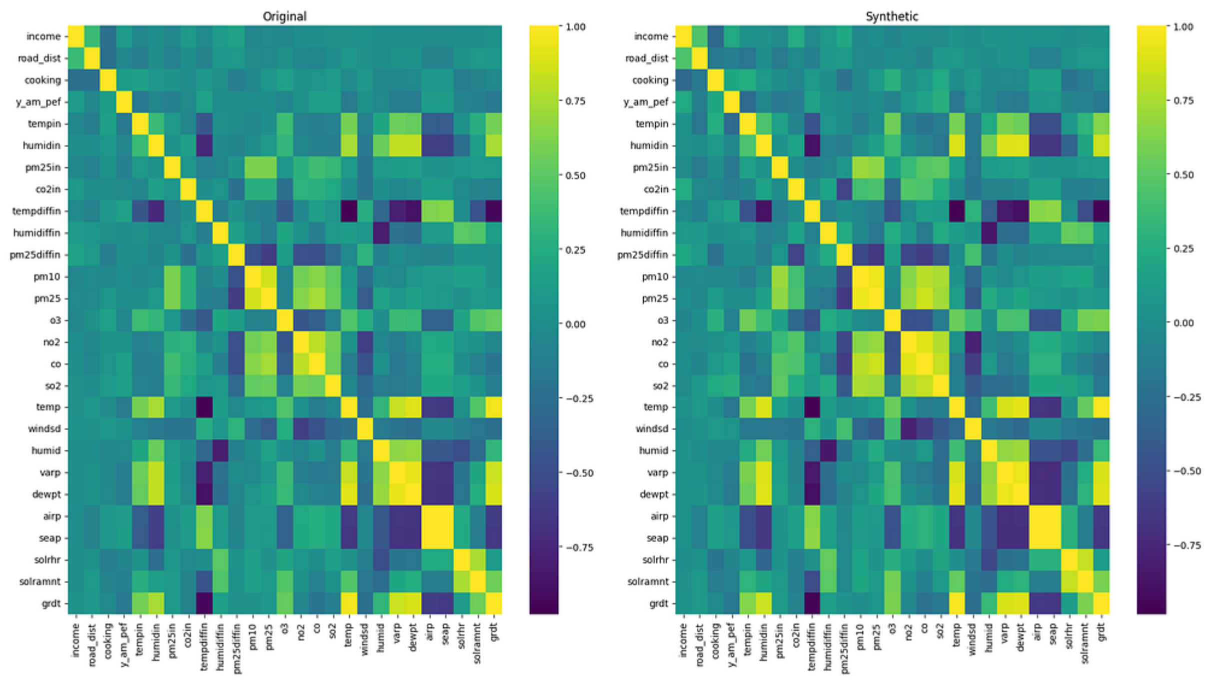
Best Single Encoder Model

Best Model Architecture: L27_E20_B16_D18_20
Total Diverging Area: 1.49481
Average KL divergence: 0.03214

Threshold	Total number of similar rows in synthetic_df	Similarity ratio in synthetic_df	Total number of similar synthetic_df rows in original_df	Similarity ratio in synthetic_df rows that are in original_df
0.1	1	0.001	0	0
0.25	5	0.004	0	0
0.5	254	0.223	20	0.009



Original vs Synthetic Data Distribution



Original vs Synthetic Data Heatmap

Balanced Approach

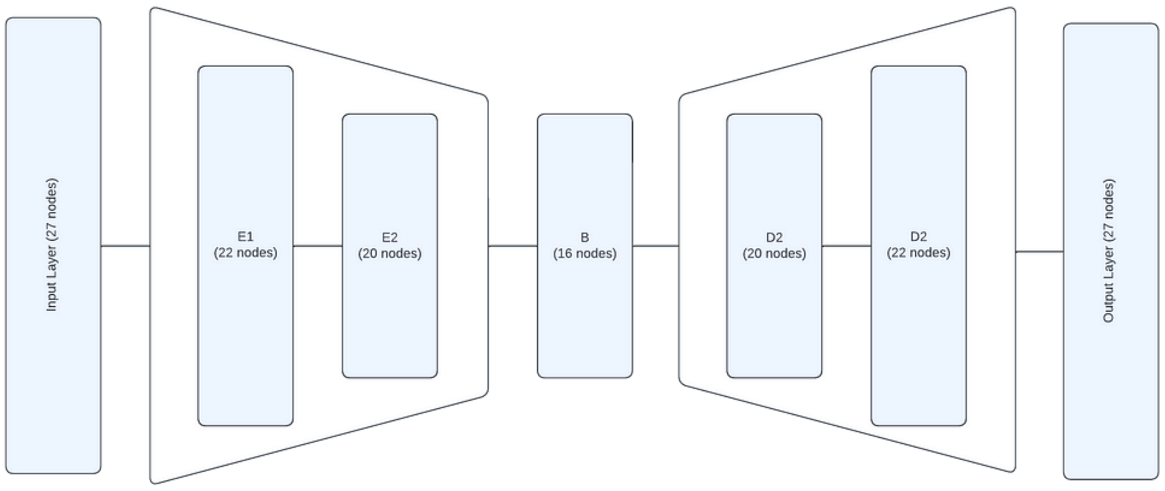
In comparison to the single-encoder model, the balanced approach demonstrated slightly lower performance. The Balanced approach introduces an additional dense layer in the encoder part of the model, facilitating the mapping of input data to a lower-dimensional latent space through a more intricate and multi-step process. Each added encoder layer progressively extracts and encodes different levels of abstraction and features from the input data.

The incorporation of the extra dense layer does increase the model's complexity by introducing more trainable parameters. In our case, where the dataset size was limited, this heightened complexity posed a challenge. Nevertheless, it's worth noting that our experiments revealed that the best-performing balanced model had the highest number of nodes in each layer and this particular model architecture proved to be an advantage and outperformed several single encoder models.

> Our experiments suggest that a balanced autoencoder with a higher number of nodes in each layer, such as the two encoding and two decoding layers with twenty-two and twenty nodes in encoding and twenty and twenty-two nodes in decoding, approached close to the performance of a single encoder model with eighteen encoding nodes.

While the balanced approach's complexity might have posed challenges with limited data, the increased expressiveness offered by additional nodes appears to be beneficial in achieving competitive results.

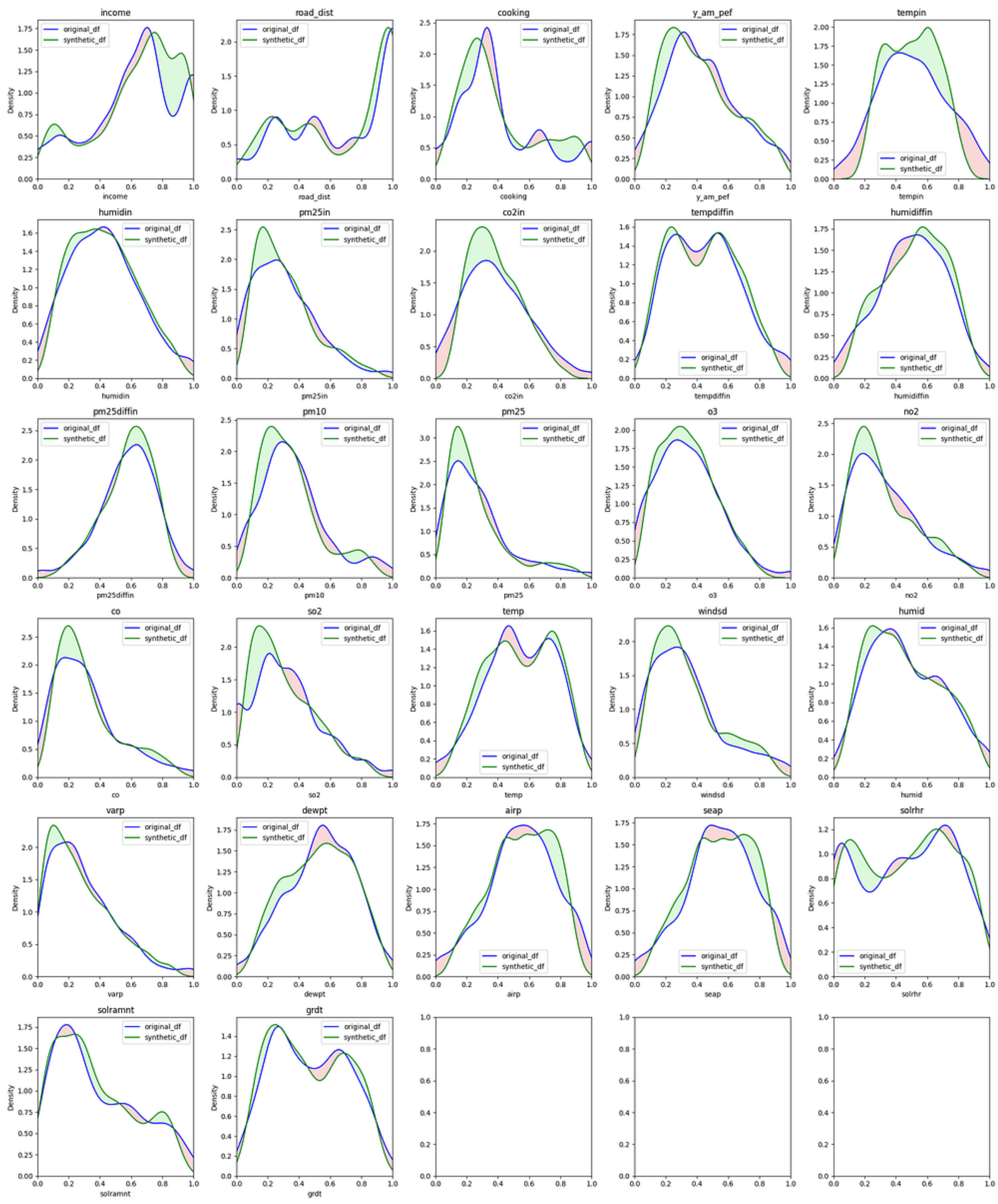
In conclusion, while the balanced approach did not surpass the best single-encoder model in our specific scenario with a limited dataset, our findings emphasize the significance of architectural choices. By experimenting with a balanced autoencoder and optimizing the number of nodes in each layer, it is possible to approach the performance of single-encoder models. Further investigation and exploration of various architectures will continue to yield valuable insights into the ever-evolving landscape of synthetic data generation.



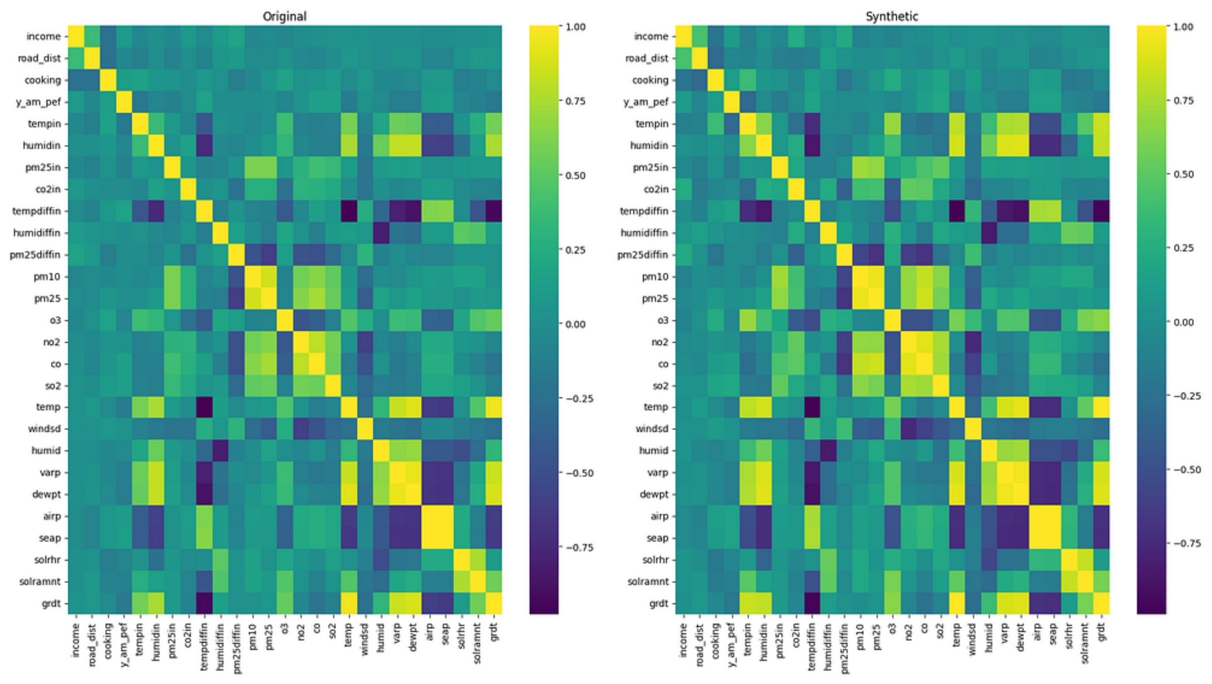
Best Balanced Model

Best Model Architecture: L27_E22_20_B16_D20_22
Total Diverging Area: 1.67873
Average KL divergence: 0.04375

Threshold	Total number of similar rows in synthetic_df	Similarity ratio in synthetic_df	Total number of similar synthetic_df rows in original_df	Similarity ratio in synthetic_df rows that are in original_df
0.1	1	0.001	0	0
0.25	3	0.003	0	0
0.5	289	0.254	24	0.011



Original vs Synthetic Data Distribution



Original vs Synthetic Data Heatmap

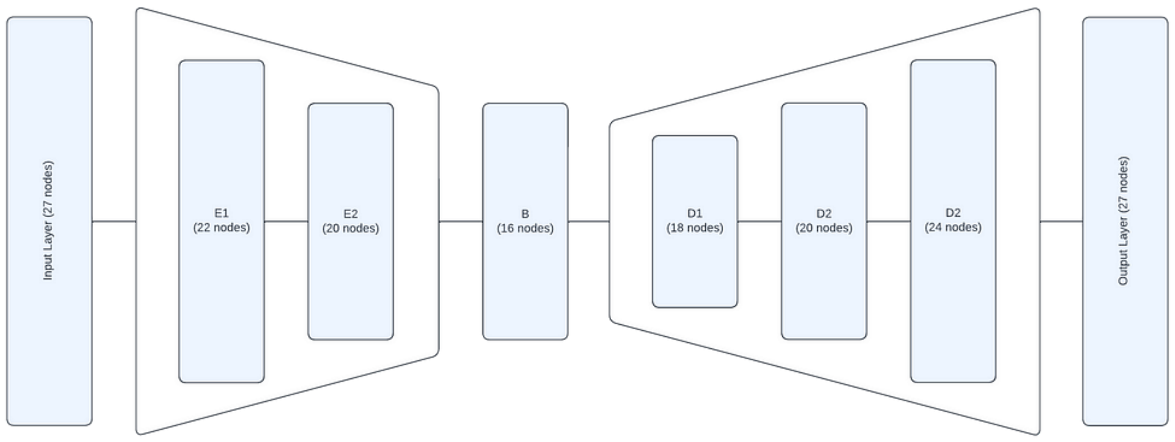
Heavy Decoder Approach

The heavy decoder approach was characterized by having a higher number of decoding layers compared to the encoding layers. This architecture aimed to progressively refine the reconstruction process, capturing intricate details and contributing to a more expressive representation of the data.

> In our experiments with a relatively smaller dataset, the heavy decoder showed signs of overfitting. The abundance of decoder layers, combined with the limited size of the dataset, led the model to memorize the training data rather than generalize.

Consequently, the heavy decoder produced duplicate and redundant results, as it struggled to effectively capture the true underlying patterns and instead memorized the data. This is evident from the Table below which highlights that even the best Heavy decoder model had the highest total number of similar rows in the synthetic data frame.

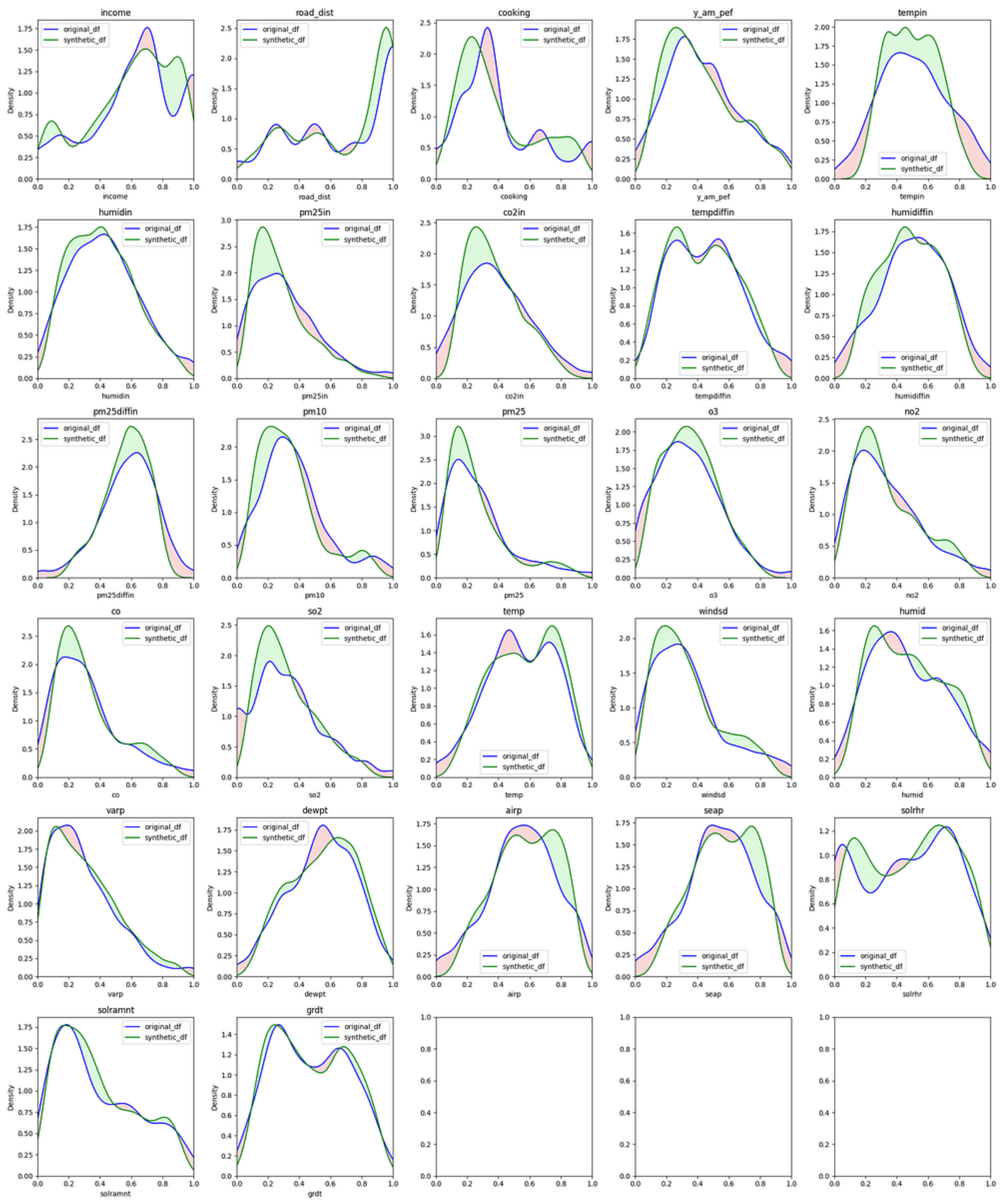
The overfitting issue highlights an important consideration when applying the heavy decoder approach to smaller datasets. With a greater number of trainable parameters and the additional decoder layers, the model's capacity to represent the limited data might exceed what is necessary, leading to a loss of generalization ability. The rest of the heavy decoder models also performed poorly and were not competitive with the single encoder or the balanced model approach.



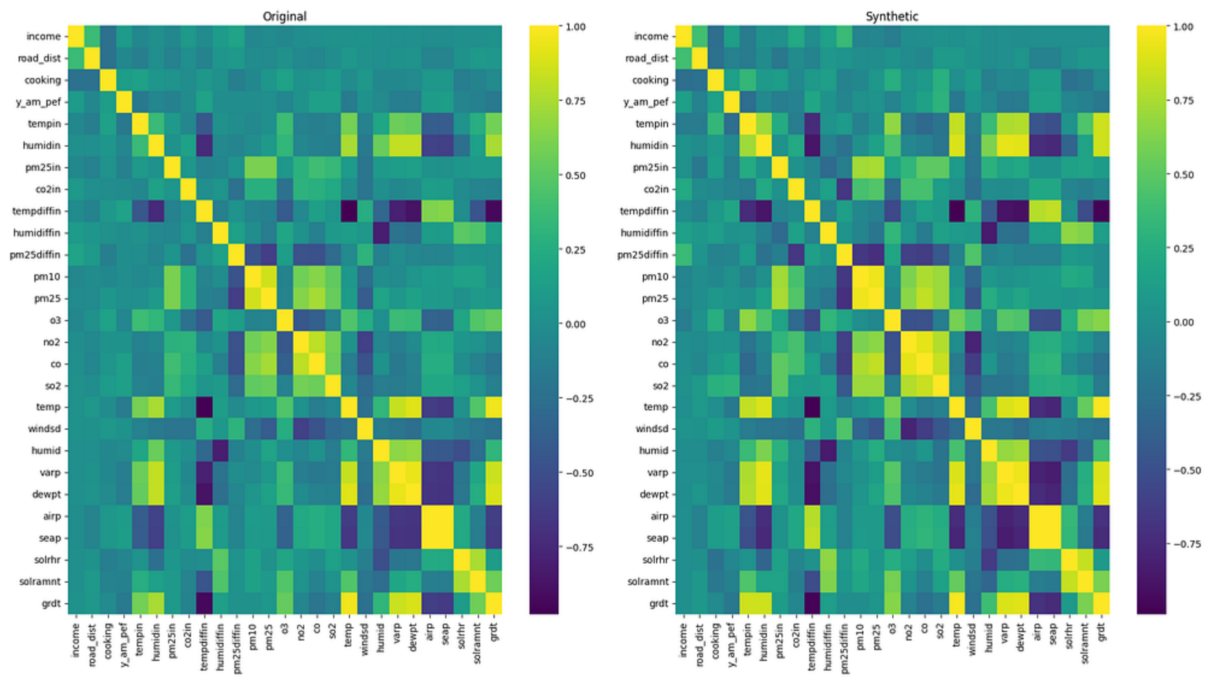
Best Heavy Decoder Model

Best Model Architecture: L27_E22_20_B16_D18_20_22_24
Total Diverging Area: 1.78775
Average KL divergence: 0.053395

Threshold	Total number of similar rows in synthetic_df	Similarity ratio in synthetic_df	Total number of similar synthetic_df rows in original_df	Similarity ratio in synthetic_df rows that are in original_df
0.1	1	0.001	0	0
0.25	6	0.005	0	0
0.5	421	0.37	20	0.009



Original vs Synthetic Data Distribution



Original vs Synthetic Data Heatmap

Conclusion

Based on the evaluation of different autoencoder models on our dataset, we have drawn valuable insights regarding their performance and suitability. The single encoder model with two decoder layers emerged as the top-performing architecture, outperforming both the heavy decoder and balanced approaches. This finding is particularly significant given the limited size of our dataset, suggesting that a simpler model can be more effective when data is scarce.

The heavy decoder approach, with its higher number of decoding layers, proved to be prone to overfitting on our relatively smaller dataset. This overfitting led to the memorization of training data and hindered the model's ability to generalize, resulting in duplicate and redundant synthetic data. Consequently, the heavy decoder approach did not perform competitively compared to the other architectures.

The balanced approach, which introduced an additional dense layer in the encoder, showed promise, especially when using a higher number of nodes in each layer. While it did not surpass the single encoder model's performance in our specific scenario, it demonstrated that architectural choices and parameter optimization can lead to competitive results. The balanced approach's increased expressiveness through additional nodes appeared beneficial for generating synthetic data.

Overall, the best model architecture for our dataset was the single encoder model with two decoder layers, denoted as L27_E20_B16_D18_20. It struck an effective balance between model complexity and data representation, resulting in superior performance compared to the other approaches. This finding highlights the importance of carefully selecting an appropriate architecture tailored to the dataset's characteristics and size.

It's essential to recognize that the performance of different models can vary depending on the dataset and its specific features. While the single encoder model was optimal for our scenario, it may not always be the case for other datasets and applications. Continual exploration and experimentation with various architectures will continue to yield valuable insights into synthetic data generation.