ADDRESSING IMBALANCED DATA IN CREDIT CARD FRAUD DETECTION USING A SYNTHETIC DATA GENERATION GAN

CÉDRIC LERESCHE AND KHALIL JALLED

Abstract. In the realm of credit card transactions, it is crucial for companies and the security of their clients to accurately identify fraudulent activities, ensuring that customers are not wrongfully charged. This project explores a dataset of European cardholder transactions from September 2013, consisting of 284,807 transactions over two days, including 492 fraudulent instances, thus presenting a highly imbalanced data representation with frauds just accounting for 0.172% of total transactions. Conventional supervised learning models may struggle with accurate classification. Therefore, we propose the application of a Generative Adversarial Network model to generate synthetic data for more balanced training of a classifier that will be in charge of fraud detection.

Key words. Credit Card Fraud Detection, Imbalanced Data, Synthetic Data Generation, Generative Adversarial Networks (GAN), Classification models, Supervised Learning, Conditional GAN, Machine Learning, Fraud Classification.

AMS subject classifications. 62H30, 62P20, 68T05, 68T10, 91G70.

1. Introduction.

The widespread use of credit cards as a convenient mode of payment has led to an unprecedented surge in digital transactions. However, this trend has been accompanied by a rise in fraudulent activities. Credit card fraud is a multibillion-dollar problem, and for financial institutions, the task of identifying illegitimate transactions amid a sea of legitimate ones poses a significant challenge. This task is further compounded by the heavily imbalanced nature of the data, with fraudulent transactions representing a minuscule fraction of the total transactions. Such imbalance is problematic because standard machine learning algorithms are often biased towards the majority class, leading to poor performance on the minority class, in this case, fraudulent transactions.

In response to this issue, we propose a sophisticated approach to handle a class imbalance in credit card fraud detection by leveraging the synthetic data generation capabilities of a Generative Adversarial Network (GAN). As a milestone in machine learning, GANs have proven effective in creating synthetic data that closely mirrors real instances. However, the training of GANs can be a tumultuous journey owing to their inherent instability. They involve a delicate min-max optimization problem requiring a well-coordinated training balance between the generator and discriminator networks. Failing to maintain this balance could result in the generator producing nonsensical outputs or the discriminator falling short in accurately evaluating the generator's outputs. This challenging training procedure, coupled with the necessity to generate realistic synthetic minority samples, prompted us to explore the Conditional GAN (cGAN), a GAN variant. Unlike traditional GANs, cGANs employ additional conditioning variables to both the generator and discriminator, providing a more controlled data generation process and making it an optimal choice for creating synthetic instances of the minority class to enrich our dataset.

Our study begins with an extensive exploration of the current literature on credit card fraud detection, concentrating on diverse approaches to handling imbalanced data and leveraging synthetic data generation techniques. Simultaneously, we delve deeper into the field of GANs, examining their potential and pitfalls, with particular emphasis on their instability during training. This comprehensive research paves the way for our proposed methodology, integrating the use of cGANs for synthetic data augmentation and a Random Forest Classifier for accurate fraud detection.

Subsequent to the methodology, we provide a thorough analysis of our dataset, comprising 284,807 credit card transactions from European cardholders during two days in September 2013, which included 492 fraudulent transactions, emphasizing the class imbalance.

We present the implementation details of our approach in a dedicated section. We then demonstrate our experiments' results, focusing on our model's performance metrics, particularly the Area Under the Precision-Recall Curve (AUPRC), a metric more insightful than the traditional Receiver Operating Characteristic (ROC) curve for imbalanced datasets.

Finally, we conclude our study by discussing the implications of the results, offering insights into the effectiveness of our proposed solution in addressing credit card fraud detection in the face of significant data imbalance.

2. Review of Literature. Our primary research question investigates how a classification model for credit card fraud detection can be effectively enhanced by utilizing a Generative Adversarial Network (GAN) to address the issue of class imbalance within the dataset. We aim to maximize the conditional probability $P(y = 1|x)$ of correctly predicting a fraudulent transaction, denoted by $y = 1$, given a credit card transaction data point, denoted by x. However, the challenge lies in the fact that the prior probability $P(y = 1)$ of a transaction being fraudulent is significantly low due to the imbalanced nature of the data, leading a naive classifier to predict the majority class most of the time, which results in poor performance on detecting fraudulent transactions. In this project, we explore the potential of employing a Conditional GAN (cGAN), a variant of GAN, to generate synthetic instances of the minority class (fraudulent transactions) in order to balance the dataset and improve the classifier's ability to identify fraud. The overarching objective is to ensure that the synthetic data x' generated by the cGAN satisfies $P(x'|y = 1) \approx P(x|y = 1)$, thus approximating a more balanced dataset, and consequently enhancing the accuracy of fraud detection.

The original Generative Adversarial Network (GAN) was proposed by Goodfellow et al. (2014) in their paper titled "Generative Adversarial Nets" [1]. This innovative approach introduced the concept of a game-theoretic scenario between two distinct models: a generator G and a discriminator D. The generator's purpose is to create synthetic samples from random noise that can deceive the discriminator into believing these samples are from the real dataset. Simultaneously, the discriminator's goal is to differentiate between samples from the actual data distribution and the synthetic samples generated by G. This interplay is formulated as a min-max optimization game where the generator and the discriminator are trained simultaneously with competing objectives. The generator aims to maximize the error rate of the discriminator, while the discriminator tries to minimize its error rate, creating a dynamic of adversarial learning. The original paper demonstrated that GANs can model complex, multi-modal data distributions and generate realistic synthetic samples, making them an attractive tool for synthetic data generation in various domains. However, the paper also acknowledged the challenges in training GANs, such as the tendency towards mode collapse, the difficulty of achieving equilibrium in the game, and the lack of an explicit measure of the quality of generated samples.

The original GAN framework, often referred to as Vanilla GANs, indeed presented considerable challenges in terms of stability during training. It took a few years and significant contributions from researchers to navigate these difficulties effectively. Two papers published in 2021, Charitou et al. [2] and Langevin et al. [3], demonstrated the successful application of GANs in the context of credit card fraud detection. Langevin

et al. harnessed the potential of GANs for data augmentation, synthesizing counterfeit instances of fraudulent transactions to balance the dataset. This innovative approach considerably bolstered the performance of a subsequent classifier. Their work also extended the capabilities of GANs by exploring the realm of transfer learning in fraud detection, which allowed a model trained on one dataset to be adapted and utilized on another. Such a feature is particularly invaluable in the field of credit card fraud detection, given the sensitive nature of data and privacy concerns, which frequently impede access to large, diverse datasets. They employed a Wasserstein GAN's (WGAN) architecture for this purpose.

Our methodology drew substantial inspiration from the framework proposed by Charitou et al., primarily due to the resemblance of their dataset to ours. Their research illustrated that, for credit card fraud detection, a blend of their methodology and a Random Forest Classifier yielded an impressive F1 score of 91.31%. Their methodology comprised an enhanced cGAN model, which integrated the feature-matching loss mechanism introduced by Salimans et al. [4], as a replacement for the standard loss function. This enhancement empowered the GAN model to evade mode collapse and maintain stable training, significantly contributing to the model's effectiveness.

3. Methodology. The focus of our project was to develop a credit card fraud detection model. To do that, we had to address the imbalanced data problem in our original dataset. We used a particular architecture of GAN models to generate synthetic fraudulent transactions and data to augment our training set.

3.1. Data augmentation using a cGAN model. Data augmentation is a strategy often employed to enhance the diversity of training data, thereby improving the performance and generalization ability of machine learning models. In the context of our project, we confront an imbalanced dataset and aim to generate synthetic samples resembling the minority class - fraudulent transactions. To this end, we harness the capabilities of generative models, specifically cGANs.

A cGAN is comprised of two core components: a generator (G) and a discriminator (D). The generator's role is to craft synthetic samples given a random noise vector z and a class label y supposed to guide the data generation process, while the discriminator is trained to discern whether a given sample is authentic (originating from the actual dataset) or counterfeit (produced by the generator). These two components engage in a two-player minimax game, represented by the following binary cross-entropy objective function:

$$\min_G \max_D V(D,G) = E_{x\sim p_{\text{data}}(x)}[logD(x|y)] + E_{z\sim p_z(z)}[log(1-D(G(z|y)))] \tag{3.1}$$

Where x symbolizes a sample from the original data, z is a sample from the noise vector, y corresponds to the class label, and $G(z|y)$ is the generated sample.

3.1.1. Feature-Matching for Stable Training. While training a cGAN, we encounter the challenge of maintaining stability and ensuring meaningful learning. To overcome this, we incorporate a technique known as feature-matching. Rather than directly optimizing the output of the discriminator, we guide the generator to emulate the expected feature representation of real data at an intermediate layer of the discriminator. This introduces a new objective function for the generator, given by:

$$\min_{G} ||\mathbb{E}_{x\sim p_{\text{data}}(x)}[f(x)] - \mathbb{E}_{z\sim p_z(z)}[f(G(z|y))]||_2^2 \tag{3.2}$$

In this equation, $f(x)$ denotes the activations on an intermediate layer of the discriminator when input x is fed. By aligning the generator's features with the actual data features, we ensure the generator crafts samples to be statistically akin to the original data.

3.1.2. Focusing on Fraudulent Transactions. With our specific objective of generating synthetic samples that mimic fraudulent transactions ($y = 1$), we tune our model to generate data under the constraints of the fraudulent class conditional distribution. The resultant objective function becomes:

$$||\mathbb{E}_{x\sim p_{\text{data}}(x|y=1)}[f(x|y=1)] - E_{z\sim p_z(z)}[f(G(z|y=1))]||_2^2 + \mathbb{E}_{x\sim p_{\text{data}}(x|y=1)}[\log(D(x|y=1))] \tag{3.3}$$

This explicit focus on the characteristics of the fraudulent transactions is critical to the successful data augmentation of our imbalanced dataset. Consequently, the synthetic fraudulent transactions generated facilitate balancing the classes, bolstering the performance of downstream classifiers.

3.1.3. Classification using a Random Forest model. We choose to couple our synthetic data generation (SDG) process with a Random Forest (RF) classifier because of its simplicity and capacity to achieve high accuracy, especially with imbalanced datasets using minimal inputs. The augmentation of fraudulent transactions generated by our SDG process in the training set is supposed to improve the performance of our classifier.

3.2. Model selection and Training.

3.2.1. Architecture and Hyperparameters Settings of the cGAN model. In the construction of our cGAN model, we opted to replicate the hyperparameters settings (Table A.1) utilized by Charitou et al. (2021). The decision to adopt this replication approach is rooted in the intricate and often unpredictable nature of GAN model behaviour. The tuning of GAN models is notoriously challenging due to their highly complex and non-convex objective function that can lead to a variety of instability issues during training. Thus, relying on the configuration that has been empirically shown to work well allowed us to mitigate potential optimization difficulties and enhanced the overall feasibility of our model training process.

3.2.2. Hyperparameters Settings of the Random Forest Classifier. The hyperparameters for our Random Forest classifier were determined through a systematic process of hyperparameter tuning. Specifically, we used Grid Search cross-validation, a common and effective method for determining the optimal hyperparameters in a model. The hyperparameters were selected from a range of possible values to optimize the average precision score of our model, and the process was carried out on multiple parallel jobs to expedite the tuning process. The tuned hyperparameters are summarized in Table A.2. It should be noted that a `random_state` of 42 was used for the reproducibility of the results. The maximum depth of the trees (`max_depth`) was set to `None`, meaning that nodes are expanded until all leaves are pure or until all leaves contain less than `min_samples_split` samples. The `n_estimators` parameter, which specifies the number of trees in the forest, was set to 200, providing a good balance between model performance and computational efficiency.

3.3. Model Evaluation. Our primary task is binary classification, distinguishing between fraudulent and non-fraudulent transactions. Given the imbalance in our dataset, the choice of evaluation metrics is crucial. Accuracy, a commonly used metric, can be misleading in this context as a model that simply classifies all transactions as non-fraudulent would still achieve high accuracy, despite failing to detect any actual fraudulent cases.

Thus, we selected more relevant metrics, including Precision, Recall, F1-score, and the Area Under the Precision-Recall Curve (AUPRC):

- **Precision:** Precision measures the proportion of true positive predictions (i.e., correctly identified fraudulent transactions) out of all positive predictions. A high precision score indicates that when our model identifies a transaction as fraudulent, it is likely to be correct.
- **Recall:** Recall calculates the proportion of true positive predictions out of all actual positives (i.e., all fraudulent transactions). A high recall score shows that our model can correctly identify a large proportion of all fraudulent transactions. This metric is particularly important in our context because failing to detect a fraudulent transaction (a false negative) can have more severe consequences than falsely flagging a transaction as fraudulent (a false positive).
- **F1-score:** The F1-score is the harmonic mean of precision and recall. It seeks to balance these two metrics and is particularly useful when dealing with imbalanced datasets. A high F1-score indicates that both the precision and recall of the model are high.
- **Average Precision (AP):** Average Precision summarizes the shape of the precision-recall curve, and it computes the average precision values for recall levels over the interval from 0 to 1. The higher the AP, the better the model, with a maximum value of 1. The AP score gives more insight into the model's performance when the class distribution is imbalanced.
- **AUPRC:** The AUPRC is the area under the precision-recall curve. The precision-recall curve plots the recall (on the x-axis) against the precision (on the y-axis) for different classification thresholds. AUPRC is useful when dealing with imbalanced datasets as it takes into account both false positives and false negatives. A model with a perfect AUPRC of 1.0 is excellent, whereas a model with an AUPRC of 0.5 performs no better than random guessing.

We also experiment with different ratios of non-fraudulent to fraudulent transactions in our training dataset to understand the impact on model performance. This includes a pure Random Forest model without synthetic data generation and several SDG + RF models trained with an increased proportion of fraudulent cases using synthetic data generation. This evaluation strategy provides a comprehensive view of how the ratio of class representation influences model performance.

3.4. Data Collection and Preprocessing. We utilized a dataset provided by Kaggle, namely, the Credit Fraud Detection dataset [5].

The data features are the results of a PCA transformation to protect sensitive information. Apart from 'Time' and 'Amount', all other features (V1, V2, ..., V28) are the principal components obtained through PCA. 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset, while 'Amount' is the transaction amount. 'Class' is the target variable.

3.4.1. Data Preprocessing.

Data Preprocessing. We first began by loading the data and examining its structure and any interesting trends. A significant task involved in our preprocessing was the scaling of the 'Time' and 'Amount' features. Since the range of values in these columns was quite different from the range of values in the other columns, we scaled these features using the RobustScaler method, which is less prone to outliers because of the interquartile range used for scaling, which mitigates the impact of extreme outliers.

Data Split. We separated the features (X) from our target variable (Y) - the 'Class' feature. Subsequently, we divided the dataset into training, validation, and test subsets for model development and evaluation. We split the dataset into training, validation, and test sets. We maintained a split of 70% data for training, 15% for validation, and 15% for testing.

Model Training. Post this, we used the CGAN to generate synthetic samples and trained a Random Forest model on the augmented training dataset. Hyperparameters of the Random Forest model were tuned using the validation set. Finally, we evaluated our models using the test set.

4. Data Description and Analysis. The dataset is primarily composed of numerical input variables, which have been anonymized and transformed using Principal Component Analysis due to confidentiality constraints. As such, the original features and detailed background information related to the data are not available. The PCA-transformed features are labelled as V1, V2, ... up to V28. In addition to the PCA features, the dataset also includes two non-transformed variables: 'Time' and 'Amount'. The response variable in our dataset is the 'Class' feature. This binary variable takes the value 1 in the case of a fraudulent transaction and 0 otherwise. The objective of our research is to build a model that can accurately predict the 'Class' of a given transaction based on the other features.

Given the data's sensitive nature, careful data cleaning and preprocessing were crucial steps before analysis. The dataset was inspected for missing or inconsistent entries, and any anomalies were addressed appropriately to ensure a robust and reliable analysis. As the data has already undergone a PCA transformation, we could not perform additional feature engineering based on domain knowledge. However, feature scaling was carried out for the 'Time' and 'Amount' features to match the scale of the PCA-transformed features.

A detailed statistical summary of each feature in the dataset has been conducted, which includes count, mean, standard deviation, minimum value, 25th percentile, median, 75th percentile, and maximum value. For a comprehensive review of the data description, refer to the tables in Appendix B.1 and for the distribution of each feature in Appendix B.1 - B.28.

Observing Figure B.29, we can see that the distribution of transaction amounts is heavily skewed towards lower amounts, indicating that most transactions involve relatively small sums of money. This pattern is consistent with typical credit card usage, where large transactions are less frequent.

As depicted in Figure B.30, the distribution of transaction times shows a specific pattern, which may hint at influences from different time zones or the natural daily rhythm of card usage. These patterns can potentially aid in spotting unusual behaviour that might be indicative of fraudulent activities.

A closer analysis of the fraudulent transactions' temporal distribution (refer to Figure B.31) reveals that fraudulent activities are likely to be time-sensitive. Notably, these transactions appear to adhere to certain periodic patterns, with a higher trans-

action volume observed in the second half of the captured time frame. However, due to the lack of information about the starting hour of the day in the original dataset, we can only conjecture that the volume of fraudulent activities might be increasing during the night.

5. Code Implementation.

5.1. Data Loading and Preprocessing. Firstly, data is loaded from a CSV file and then 'Time' and 'Amount' features are scaled using the RobustScaler from the sklearn library. This is to bring all features into a similar range and mitigate the impact of outliers. The original dataset is then split into training, validation, and test sets using the train_test_split function from the sklearn library.

5.2. cGAN Model Training. The implementation of the cGAN was mainly done through the Keras library.

Here are the classes used in the code:

1. **Generator:** This class is used to create a generator model. The generator takes in noise and labels as input, combines them, and generates fake transactions. The noise is a latent random variable, and the label is the condition we impose on the generator (here the class of the fraudulent transactions). The generator consists of Dense layers followed by Batch Normalization layers. The output of the generator is a fake transaction with the same dimensions as a real transaction.
2. **Discriminator:** This class is used to create a discriminator model. The discriminator takes in real or fake transactions and labels as inputs, and determines whether the input transaction is real or fake. It consists of Dense layers, LeakyReLU layers, and Dropout layers. The output of the discriminator is a probability indicating whether the input transaction is real or fake. This class also has a feature output model for feature matching during the training process. The discriminator's model is trained on batches of real and fake transactions and labels.
3. **CGAN:** This class combines the generator and the discriminator to form the CGAN model. It has methods to build the CGAN model, compile the models, and train the CGAN. During training, real transactions and labels are sampled, fake transactions and labels are generated, and the discriminator and generator are trained in turn. Feature matching is used to help the generator generate more realistic transactions by minimizing the absolute difference between the features of real and fake transactions. This class also has a method to generate fake transactions using the trained generator.

The overall flow of the CGAN training process can be described as follows:

1. Randomly select a batch of real transactions and their labels.
2. Generate a batch of fake transactions using the generator, along with random labels.
3. Train the discriminator on the real and fake transactions and labels.
4. Use feature matching to help the generator create more realistic transactions.
5. Train the generator using the discriminator's feedback.

This process is repeated for a specified number of epochs, improving the generator's ability to create fake transactions that the discriminator cannot distinguish from real ones.

The architecture of the Generator involves the following:

```
def build_model(self):
```

```
    noise = Input(shape=(self.latent_dim,))
    label = Input(shape=(self.label_dim,))
    model_input = Concatenate()([noise, label])

    x = Dense(128, activation='relu')(model_input)
    x = Dense(64, activation='relu')(x)
    x = BatchNormalization()(x)
    x = Dense(32, activation='relu')(x)
    x = BatchNormalization()(x)
    model_output = Dense(self.output_dim, activation='tanh')(x)

    return Model([noise, label], model_output)
```

Input Layers. Take two inputs. A noise vector of size *latent_dim*, which is the random noise used by the Generator to produce the fake inputs. A label input of size *label_dim*, specifies the class of the synthetic data.

Concentration Layer. This layer concatenates the noise and label inputs using the Concatenate() function, allowing the Generator to produce an output based on the provided label.

Dense Layers. After the concatenation, the inputs pass through a series of dense layers. These layers can learn the non-linear transformations of the input data thanks to a fully connected neural network, where each neuron is connected to every neuron from the previous layer.

Batch Normalization Layers. These layers normalize the output from the activation function of the previous layer, leading to stabilization and speed-up of the learning process.

Output Layer. Another Dense layer with size *output_dim*, which uses the hyperbolic tangent function. This choice ensures that the scale of the output data matches that of the preprocessed input data.

The architecture of the Discriminator involves the following:

```
def build_model(self):
        data = Input(shape=(self.input_dim,))
        label = Input(shape=(self.label_dim,))
        model_input = Concatenate()([data, label])

        x = Dense(128)(model_input)
        x = LeakyReLU(alpha=0.2)(x)
        x = Dropout(0.2)(x)
        x = Dense(64)(x)
        x = LeakyReLU(alpha=0.2)(x)
        x = Dropout(0.2)(x)
        x = Dense(32)(x)
        feature_output = LeakyReLU(alpha=0.2)(x)
        x = Dropout(0.2)(feature_output)

        model_output = Dense(1, activation='sigmoid')(x)
        return Model([data, label], model_output), Model([data, label], feature_output)
```

Input Layer. This layer takes in a data sample with size *input_dim* and another one with size *label_dim*.

Concentration Layer. This layer concatenates the data and label together, performing the same task as the generator layer.

Dense and LeakyReLu Layers. Once concatenated, the input is passed through a series of Dense Layers followed by LeakyReLu activation functions. It is important to note that the LeakyReLu function is similar to the standard ReLu function, except that it allows small negative values when the input is below zero. This property helps prevent neurons from dying out during training.

Dropout Layers. These layers are implemented after each LeakyReLu layer. The dropout technique is used as a regularization technique, randomly setting a proportion of input units to 0 during each update of the training process. This helps prevent overfitting.

Output Layer. This is the final Dense layer that uses a Sigmoid activation function to classify the input as either real or fake.

Another model in the Discriminator function is also implemented which is the feature extractor. Indeed, the output of this model is used in the feature-matching technique during the training of the generator.

5.3. Random Forest Classifier training. We used the sklearn library to train our classifier.

The RandomForestModel class is utilized to encapsulate the Random Forest model, its training, hyperparameter tuning, evaluation, and the plotting of the precision-recall curve. Here are the methods of this class:

1. **calculate_AUPRC:** This method calculates the Area Under the Precision-Recall Curve (AUPRC) and the average precision of the model. It takes the ground truth labels and predicted scores as input and returns the average precision and AUPRC.
2. **evaluate:** This method evaluates the model by predicting probabilities on the test set, calculating the average precision and AUPRC, and plotting the precision-recall curve.
3. **tune_hyperparameters:** This method is used to perform grid search cross-validation to find the best hyperparameters for the Random Forest model. It takes the training data and a grid of hyperparameters as input. It fits the model on the training data for every combination of hyperparameters and finds the best parameters that give the highest average precision score. The model is then set to be the best estimator found by the grid search.
4. **train:** This method is used to train the model on the given data with the best parameters found during hyperparameter tuning.
5. **plot_AUPRC:** This method plots the precision-recall curve for the given ground truth labels and predicted scores. The area under the curve is filled to visually represent the AUPRC.

The overall process of training the Random Forest classifier can be described as follows:

1. Hyperparameters are tuned using grid search cross-validation on the training data.
2. The model is trained on the training data with the best parameters.
3. The model is evaluated on the test data, calculating the average precision and AUPRC.
4. The precision-recall curve is plotted, showing the trade-off between precision and recall for different thresholds.

This process results in a trained Random Forest classifier with tuned hyperparameters, and the evaluation results provide an understanding of the model's performance.

5.4. Model Evaluation and Comparison. Model performance metrics are evaluated with and without synthetic data generation for different ratios of non-fraudulent to fraudulent transactions in the training set. Based on the desired ratio of non-fraudulent to fraudulent transactions, the number of synthetic fraudulent transactions needed is calculated. These synthetic transactions are then generated using the trained cGAN.

```
num_majority = sum(y_train == 0)
num_minority = sum(y_train == 1)
desired_ratio = 2  # ratio of non-fraudulent to fraudulent
num_samples = calculate_num_samples(num_majority, num_minority, desired_ratio)
synthetic_data = cgan.generate_samples(num_samples)
```

5.5. Training and Evaluation with Synthetic Data. The original and synthetic transactions are combined and used to train a Random Forest classifier. This classifier is then evaluated on the test data.

```
X_train_combined = pd.concat([X_train, synthetic_data])
y_train_combined = pd.concat([y_train, synthetic_labels])
rf_model.train(X_train_combined, y_train_combined)
rf_model.evaluate(X_test, y_test)
```

6. Results. We compared the performance of a Random Forest (RF) model and Synthetic Data Generation (SDG) coupled with RF with different non-fraudulent to fraudulent ratios. The comparison metrics include Precision, Recall, F1-score, Average precision, and Area Under the Precision-Recall Curve (AUPRC).

The results of the comparison are presented in Table C.1.

6.1. Discussion. Analyzing the performance metrics of our models reveals some key findings. The standalone Rf model exhibits robust performance across all metrics, including precision, recall, F1-score, average precision, and Area Under the Precision-Recall Curve (AUPRC). However, the Synthetic Data Generation with Random Forest (SDG-RF) models, particularly those with ratios of 0.5 and 1, show comparable performance. The advantages of these models become more evident when we consider precision, a crucial metric in fraud detection. Higher precision means fewer legitimate transactions are incorrectly flagged as fraudulent, reducing potential disruptions to the user experience. Both SDG_RF_0.5 and SDG_RF_1 achieve a precision of 0.94, which is slightly better than the RF model. Even though these models have a lower F1-score and recall than the RF model, their relatively high precision and comparable AUPRC metrics indicate a balance between precision and recall that still holds strong potential for fraud detection. Furthermore, even the models with lower synthetic-to-real data ratios (SDG_RF_0.01, SDG_RF_0.05, SDG_RF_0.1, SDG_RF_0.25) exhibit commendable performance. This performance suggests the value of augmenting real data with synthetic data, even in smaller proportions.

The robust performance of the standalone Random Forest model (RF) on our imbalanced dataset is due to its intrinsic features. RF builds an ensemble of decision trees, each trained on a distinct data subset, and assigns the class that appears most frequently in individual trees. This ensemble approach naturally mitigates the challenges of imbalanced data by focusing on distinguishing features that effectively partition the classes, hence reducing the pervasive bias towards the majority class. Unlike the RF model, our cGAN model wasn't subject to fine-tuning on a validation set. The RF model's performance was optimized by refining hyperparameters, which could be one reason for its superior performance. However, it's crucial to note that

with an appropriately fine-tuned cGAN, we could expect even more competitive, if not superior, performance to the standalone RF. It suggests that cGAN tuning could yield significant improvements in handling imbalanced data in fraud detection.

In terms of synthetic data, higher ratios of fraudulent transactions in the training set resulted in better performance. This can be ascribed to the mitigation of data imbalance inherent in fraud detection tasks. By augmenting real data with synthetic fraudulent transactions, the model gets an opportunity to learn the characteristics of the 'fraudulent' class more effectively, thus improving its overall performance. For the SDG-RF models with smaller ratios (0.01, 0.05, 0.1, and 0.25), although their performance was slightly lower compared to the standalone RF, they still demonstrated competitive results. This suggests that even a smaller proportion of synthetic data could be beneficial in improving model performance.

6.2. Evaluation of Synthetic Data Quality. As presented in Table C.2, substantial disparities exist between the statistical properties of the real and synthetic fraudulent transactions across all features. Predominantly, the synthetic data appears to possess smaller means and standard deviations compared to their real counterparts, with most features gravitating towards a mean near zero.

A notable case is feature V3; the real data showcases a mean and standard deviation of -6.69 and 6.83, respectively, while the synthetic variant presents significantly condensed values of -0.18 and 0.32, respectively. This observation suggests a propensity for synthetic data distributions to be more densely congregated around the mean as opposed to the real data – a pattern consistently observable across most features.

Despite the aforementioned pattern, certain features like V11 and V21 manifest higher mean values within synthetic data compared to the real data. This discrepancy could signal potential biases embedded within the synthetic data generation process. Furthermore, the reduced standard deviations observed in most synthetic data features denote a diminished variability compared to real data. This constriction could introduce challenges when employing synthetic data to train machine learning models, as they might fail to adequately capture the diverse variability inherent in real fraudulent transactions.

These conclusions are graphically corroborated by Figures C.1-C.59, which juxtapose the distributions of real and synthetic fraudulent transactions for each feature. Although the figures display certain resemblances, significant disparities, particularly in standard deviation, are glaringly evident.

However, a notable counterpoint to this argument can be seen in Figures C.2-C.60, which contrast the distributions of the augmented (real + synthetic) training set against the validation and test sets. Here, the composite distribution of the augmented training set is seen to be closely aligned with the distribution of real data.

7. Conclusion. This study evaluated the potential of synthetic data augmentation to enhance the performance of machine learning models, specifically in fraudulent transaction detection. The experimentation with a Random Forest (RF) model, trained on synthetic data, produced comparable results to that trained solely on real data, underlining the potential benefits of incorporating synthetic data.

However, a noteworthy disparity in the statistical properties between the real and synthetic data was identified, alluding to potential biases introduced during the synthetic data generation process. Despite these challenges, the concept of enriching real data with synthetic counterparts showed considerable promise, particularly in scenarios where real data is either limited or costly to acquire.

Future research will delve into the use of diverse Generative Adversarial Networks,

fine-tuning of model hyperparameters, and inclusion of additional predictive features, all aimed at achieving a more authentic replication of the real data's complexity and variability. This refined approach could pave the way for the development of more accurate and robust models for fraud detection. Additionally, we intend to explore the creation of fine-tuned models capable of generating both fraudulent and non-fraudulent transactions. This will potentially bolster the predictive power of fraud detection systems.

Appendix A. Model selection and Training.

Hyperparameter	**Value**
Optimizer	Adam
Learning Rate	0.0001
Beta 1	0.5
Epochs	100
Batch Size	64
Generator Hidden Layers	3 (128, 64, 32 neurons)
Generator Activation	ReLU and Tanh
Discriminator Hidden Layers	3 (128, 64, 32 neurons)
Discriminator Activation	LeakyReLU and Sigmoid
Latent Dimension	50
Latent Distribution	Normal (mean=0, standard deviation=1)
Input Dimension	Dimensions of Training Data
Label Dimension	1

Table A.1: Hyperparameters and architecture of the cGAN model.

Hyperparameter	**Value**
max_depth	None
min_samples_leaf	1
min_samples_split	5
n_estimators	200
random_state	42

Table A.2: Hyperparameters of the Random Forest classifier

Appendix B. Data Description and Analysis.

	Time	V1	V2	V3	V4	V5
count	284807.000000	284807.000000	284807.000000	284807.000000	284807.000000	284807.000000
mean	94813.859575	0.000000	0.000000	-0.000000	0.000000	0.000000
std	47488.145955	1.958696	1.651309	1.516255	1.415869	1.380247
min	0.000000	-56.407510	-72.715728	-48.325589	-5.683171	-113.743307
25%	54201.500000	-0.920373	-0.598550	-0.890365	-0.848640	-0.691597
50%	84692.000000	0.018109	0.065486	0.179846	-0.019847	-0.054336
75%	139320.500000	1.315642	0.803724	1.027196	0.743341	0.611926
max	172792.000000	2.454930	22.057729	9.382558	16.875344	34.801666

	V6	V7	V8	V9	V10
count	284807.000000	284807.000000	284807.000000	284807.000000	284807.000000
mean	0.000000	-0.000000	0.000000	-0.000000	0.000000
std	1.332271	1.237094	1.194353	1.098632	1.088850
min	-26.160506	-43.557242	-73.216718	-13.434066	-24.588262
25%	-0.768296	-0.554076	-0.208630	-0.643098	-0.535426
50%	-0.274187	0.040103	0.022358	-0.051429	-0.092917
75%	0.398565	0.570436	0.327346	0.597139	0.453923
max	73.301626	120.589494	20.007208	15.594995	23.745136

	V11	V12	V13	V14	V15
count	284807.000000	284807.000000	284807.000000	284807.000000	284807.000000
mean	0.000000	-0.000000	0.000000	0.000000	0.000000
std	1.020713	0.999201	0.995274	0.958596	0.915316
min	-4.797473	-18.683715	-5.791881	-19.214325	-4.498945
25%	-0.762494	-0.405571	-0.648539	-0.425574	-0.582884
50%	-0.032757	0.140033	-0.013568	0.050601	0.048072
75%	0.739593	0.618238	0.662505	0.493150	0.648821
max	12.018913	7.848392	7.126883	10.526766	8.877742

	V16	V17	V18	V19	V20
count	284807.000000	284807.000000	284807.000000	284807.000000	284807.000000
mean	0.000000	-0.000000	0.000000	0.000000	0.000000
std	0.876253	0.849337	0.838176	0.814041	0.770925
min	-14.129855	-25.162799	-9.498746	-7.213527	-54.497720
25%	-0.468037	-0.483748	-0.498850	-0.456299	-0.211721
50%	0.066413	-0.065676	-0.003636	0.003735	-0.062481
75%	0.523296	0.399675	0.500807	0.458949	0.133041
max	17.315112	9.253526	5.041069	5.591971	39.420904

	V21	V22	V23	V24	V25
count	284807.000000	284807.000000	284807.000000	284807.000000	284807.000000
mean	0.000000	-0.000000	0.000000	0.000000	0.000000
std	0.734524	0.725702	0.624460	0.605647	0.521278
min	-34.830382	-10.933144	-44.807735	-2.836627	-10.295397
25%	-0.228395	-0.542350	-0.161846	-0.354586	-0.317145
50%	-0.029450	0.006782	-0.011193	0.040976	0.016594
75%	0.186377	0.528554	0.147642	0.439527	0.350716
max	27.202839	10.503090	22.528412	4.584549	7.519589

	V26	V27	V28	Amount	Class
count	284807.000000	284807.000000	284807.000000	284807.000000	284807.000000
mean	0.000000	-0.000000	-0.000000	88.349619	0.001727
std	0.482227	0.403632	0.330083	250.120109	0.041527
min	-2.604551	-22.565679	-15.430084	0.000000	0.000000
25%	-0.326984	-0.070840	-0.052960	5.600000	0.000000
50%	-0.052139	0.001342	0.011244	22.000000	0.000000
75%	0.240952	0.091045	0.078280	77.165000	0.000000
max	3.517346	31.612198	33.847808	25691.160000	1.000000

Table B.1: Summary statistics of the original dataset

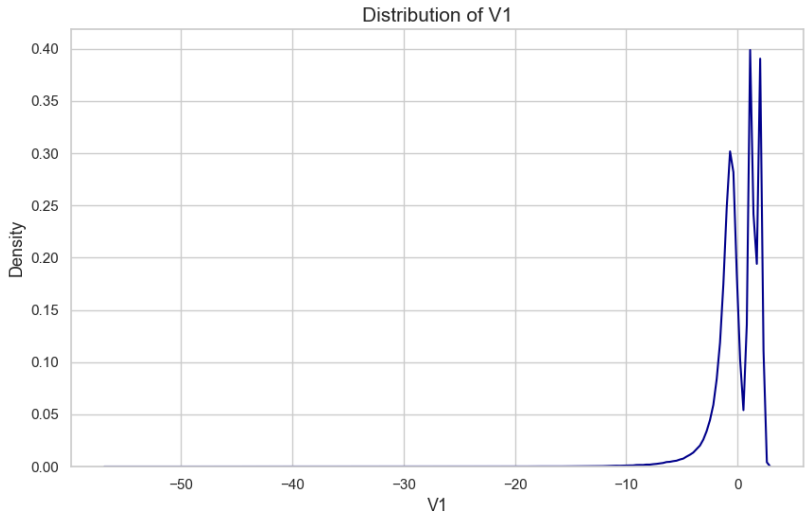

Fig. B.1: Density distribution of the feature V1.

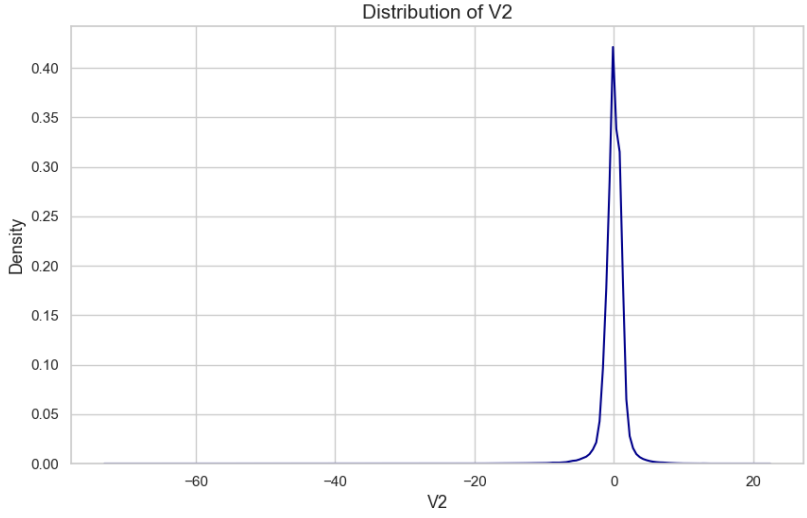

Fig. B.2: Density distribution of the feature V2.

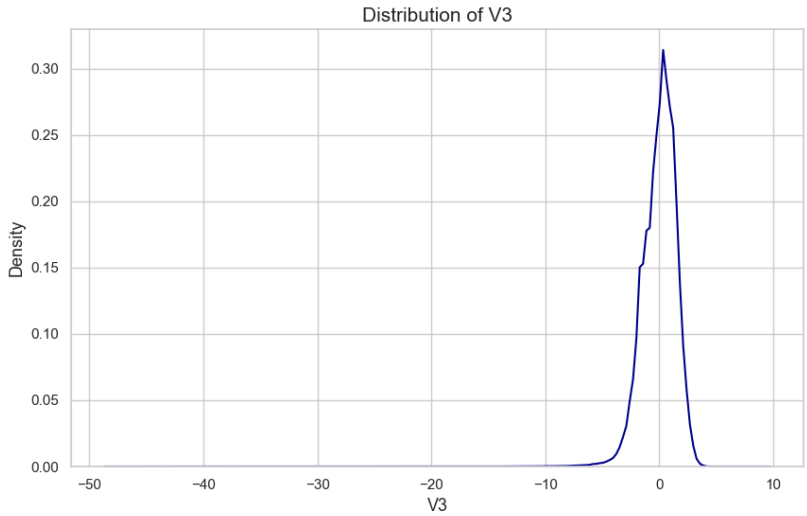

Fig. B.3: Density distribution of the feature V3.

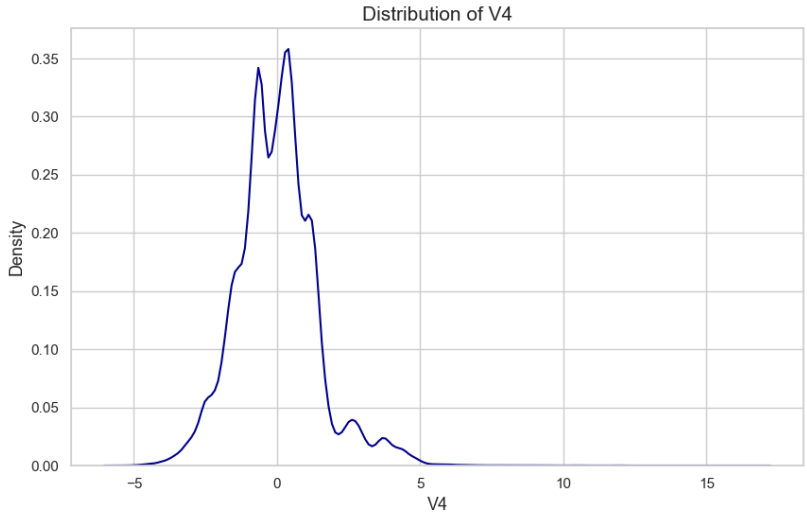

Fig. B.4: Density distribution of the feature V4.

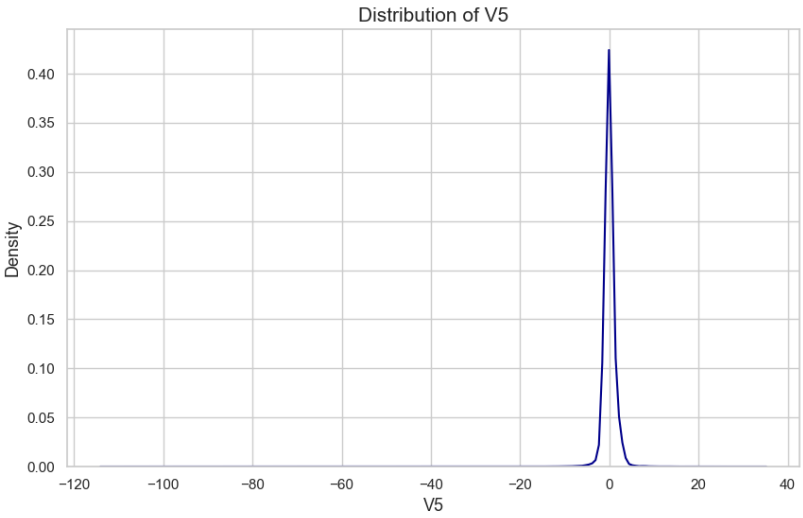

Fig. B.5: Density distribution of the feature V5.

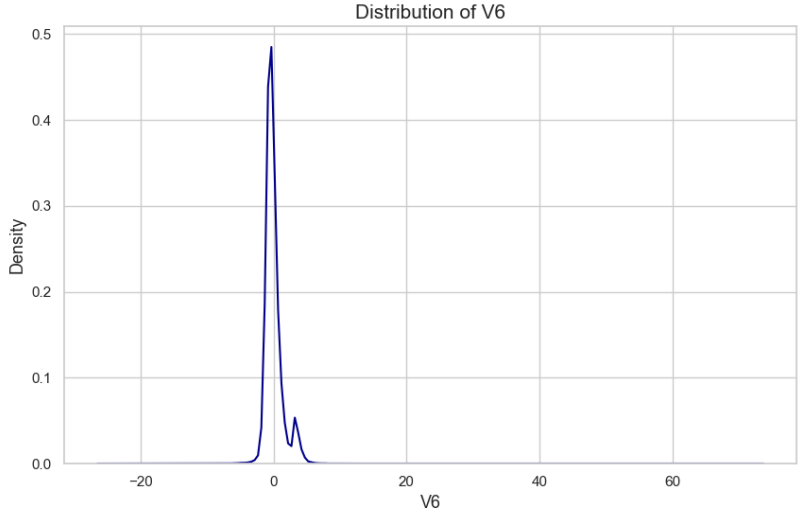

Fig. B.6: Density distribution of the feature V6.

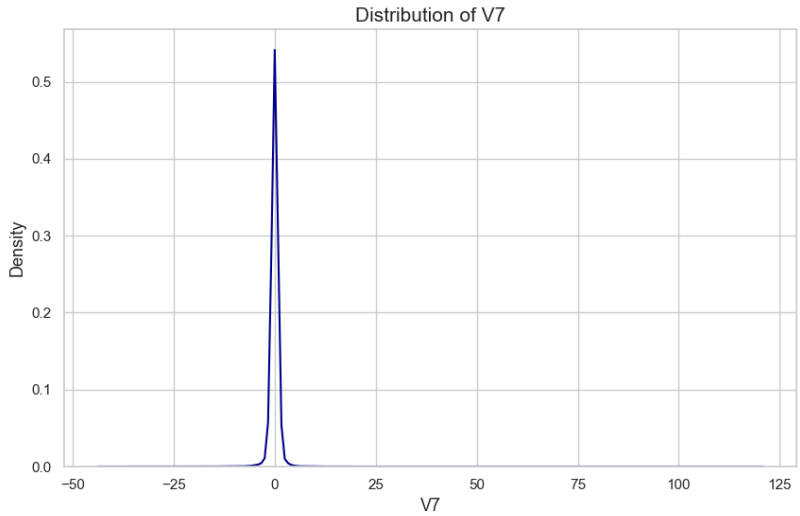

Fig. B.7: Density distribution of the feature V7.

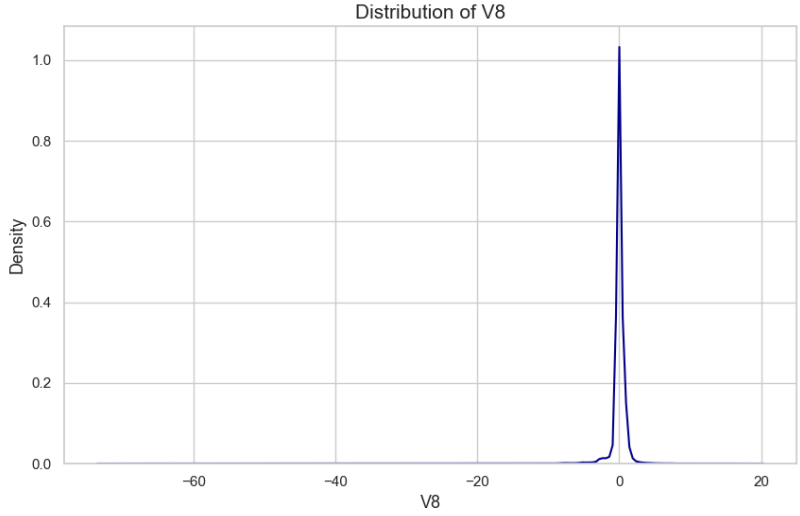

Fig. B.8: Density distribution of the feature V8.

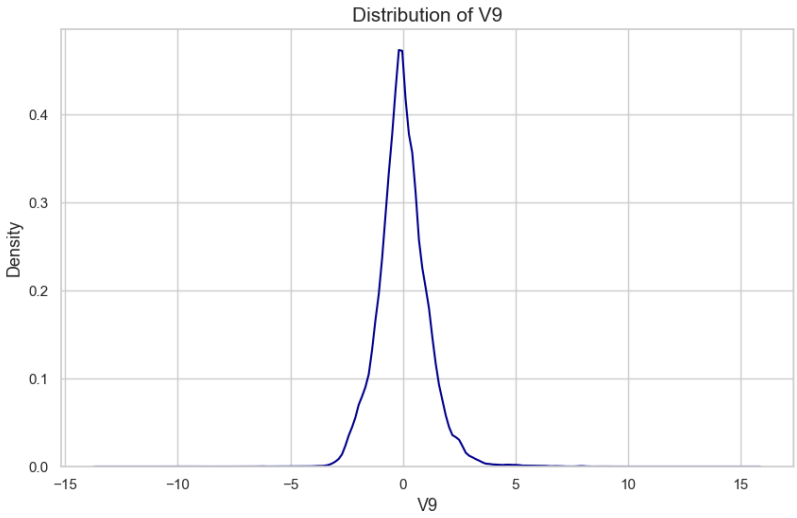

Fig. B.9: Density distribution of the feature V9.

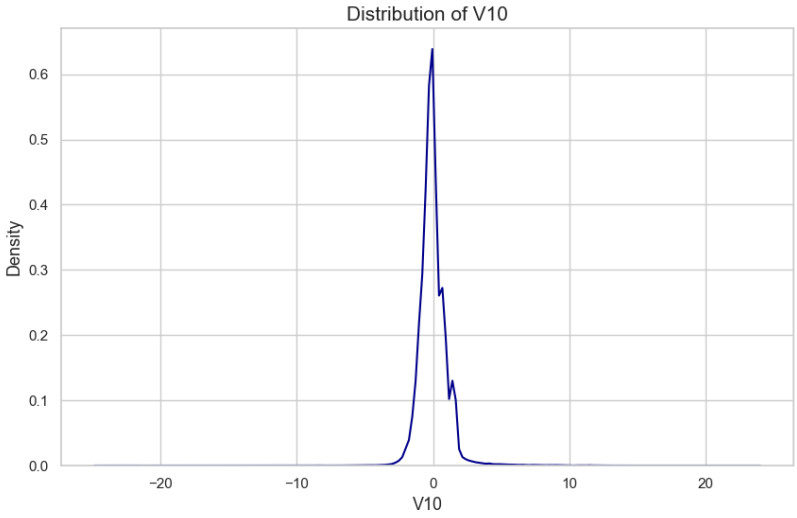

Fig. B.10: Density distribution of the feature V10.

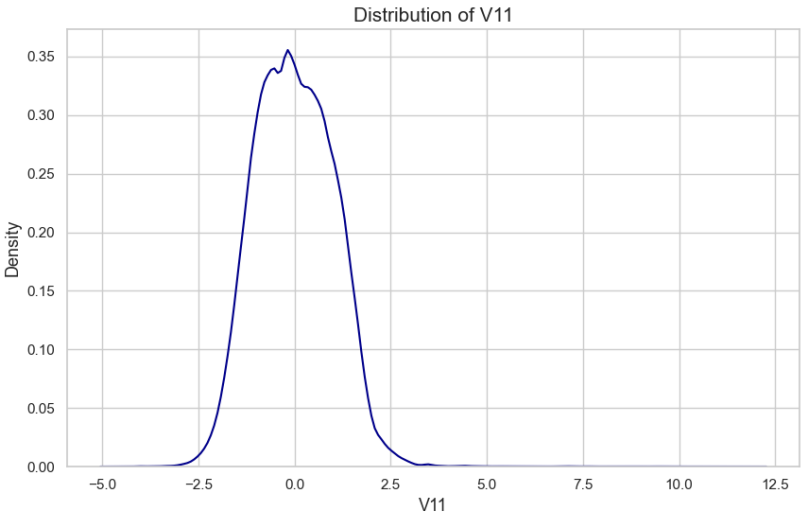

Fig. B.11: Density distribution of the feature V11.

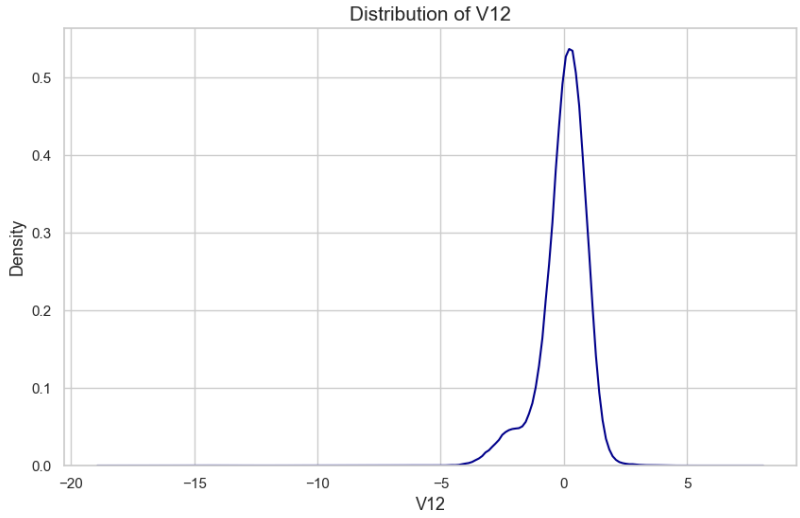

Fig. B.12: Density distribution of the feature V12.

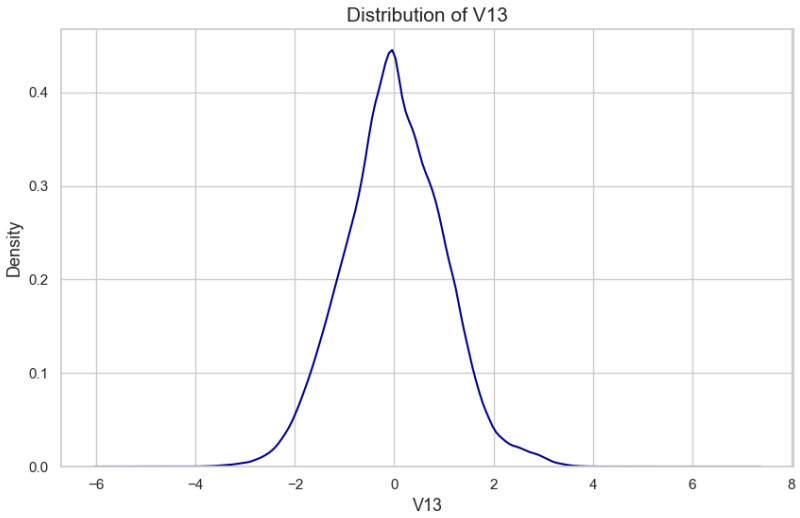

Fig. B.13: Density distribution of the feature V13.

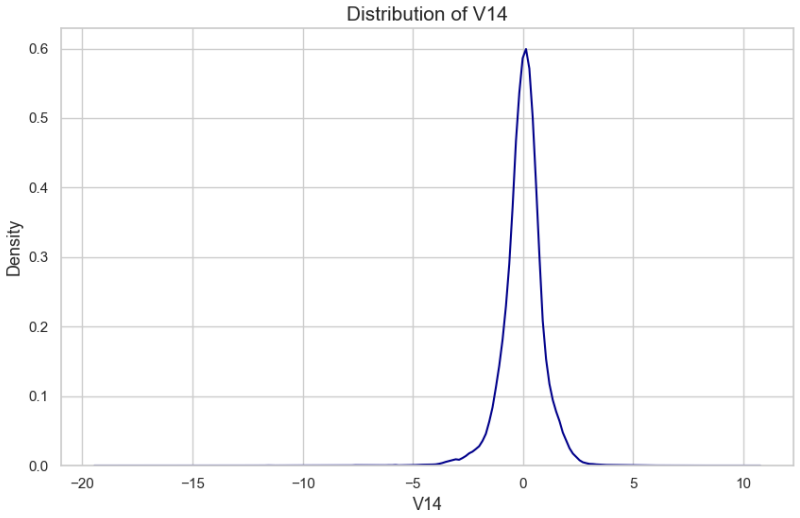

Fig. B.14: Density distribution of the feature V14.

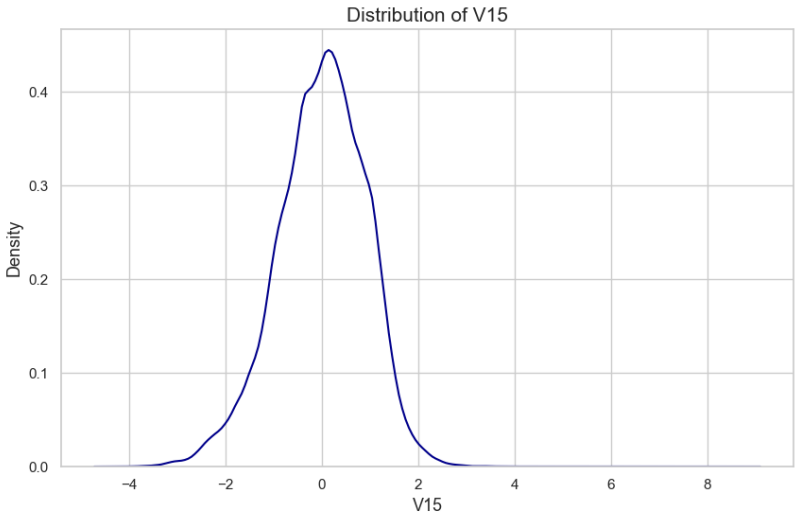

Fig. B.15: Density distribution of the feature V15.

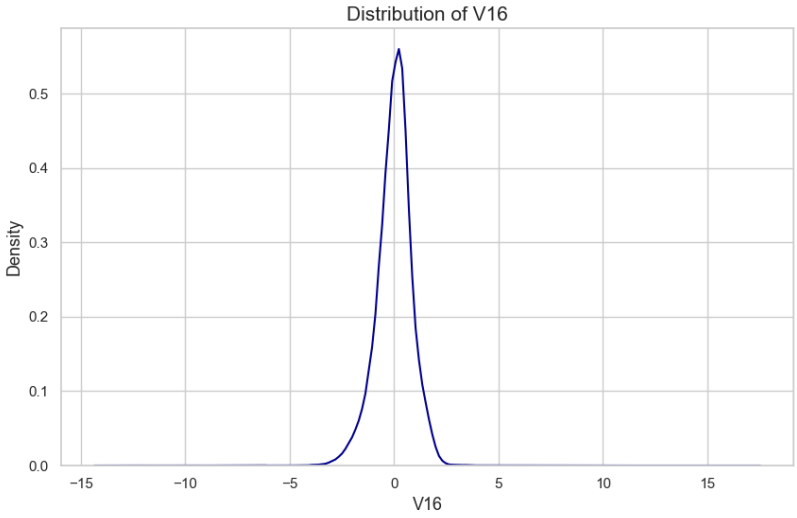

Fig. B.16: Density distribution of the feature V16.

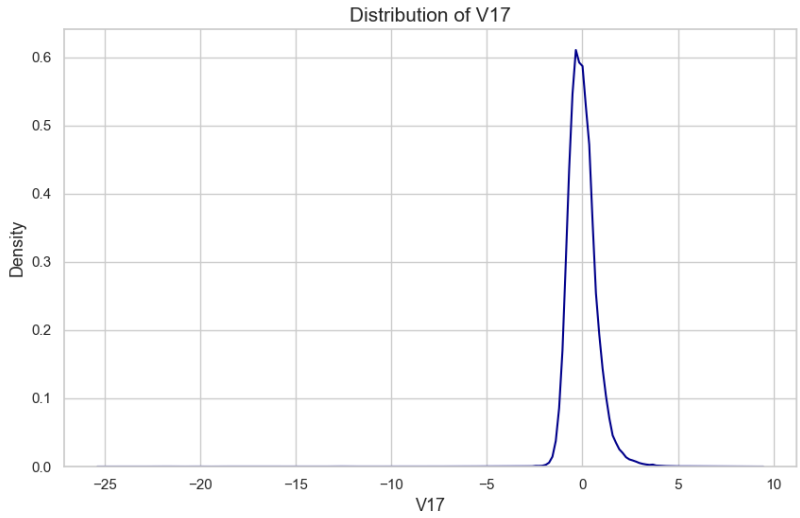

Fig. B.17: Density distribution of the feature V17.

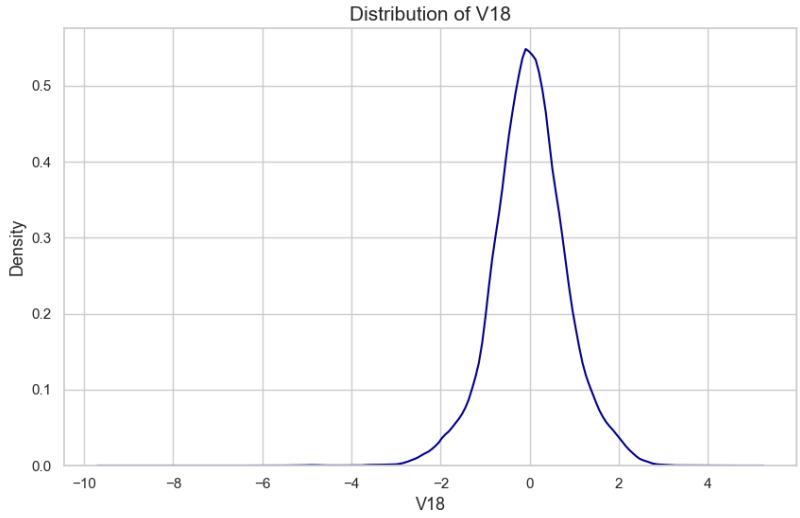

Fig. B.18: Density distribution of the feature V18.

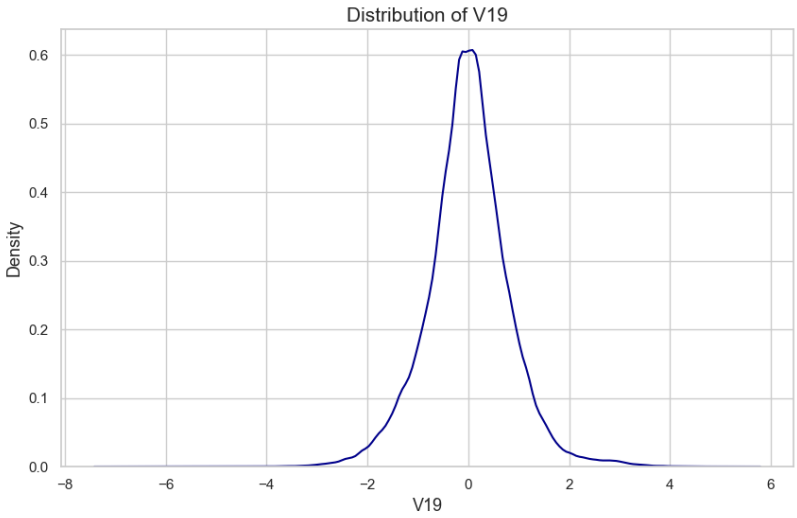

Fig. B.19: Density distribution of the feature V19.

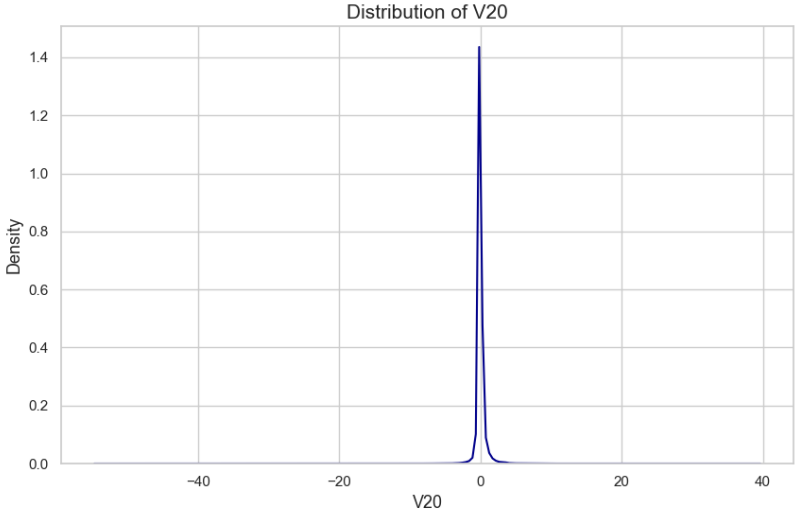

Fig. B.20: Density distribution of the feature V20.

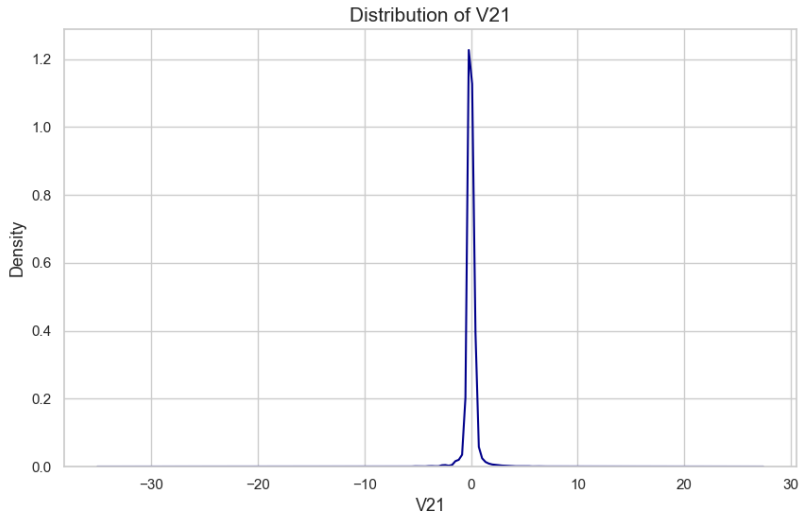

Fig. B.21: Density distribution of the feature V21.

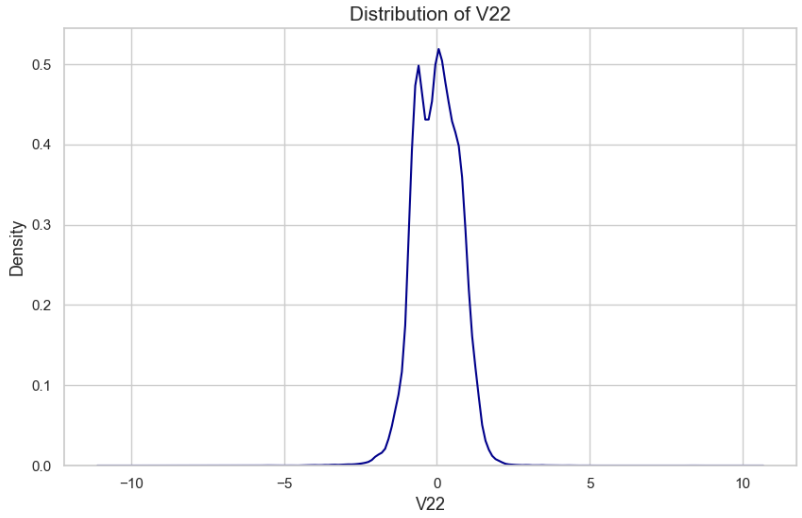

Fig. B.22: Density distribution of the feature V22.

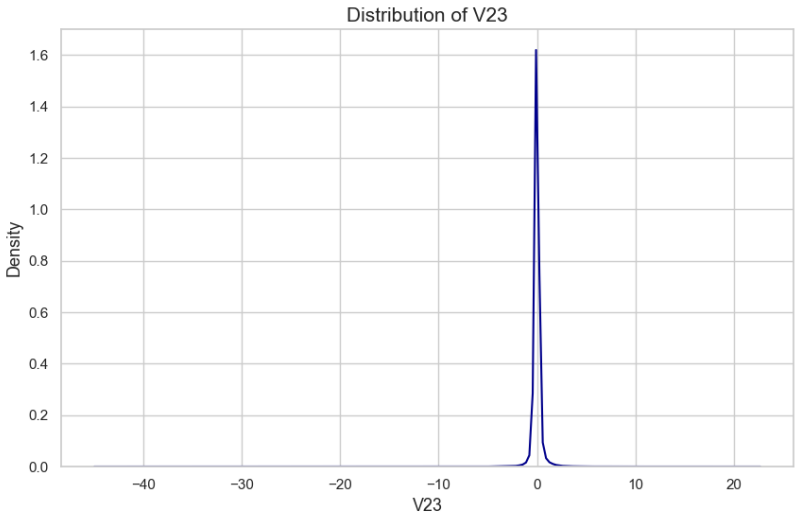

Fig. B.23: Density distribution of the feature V23.

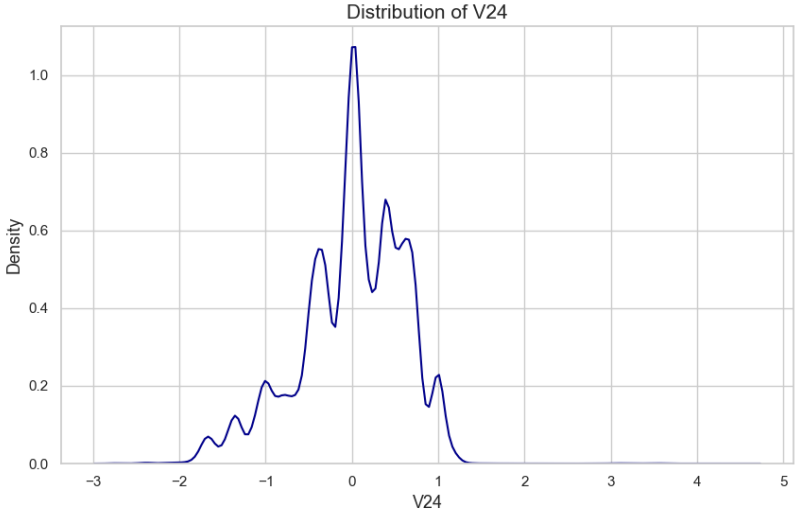

Fig. B.24: Density distribution of the feature V24.

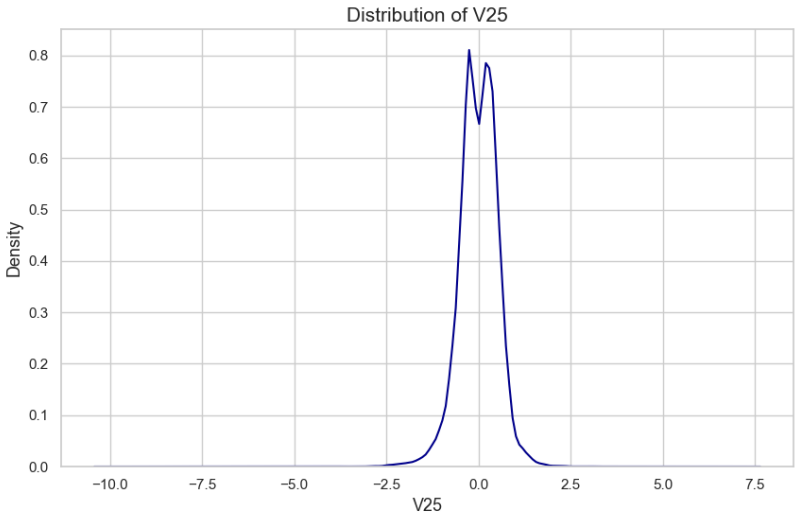

Fig. B.25: Density distribution of the feature V25.

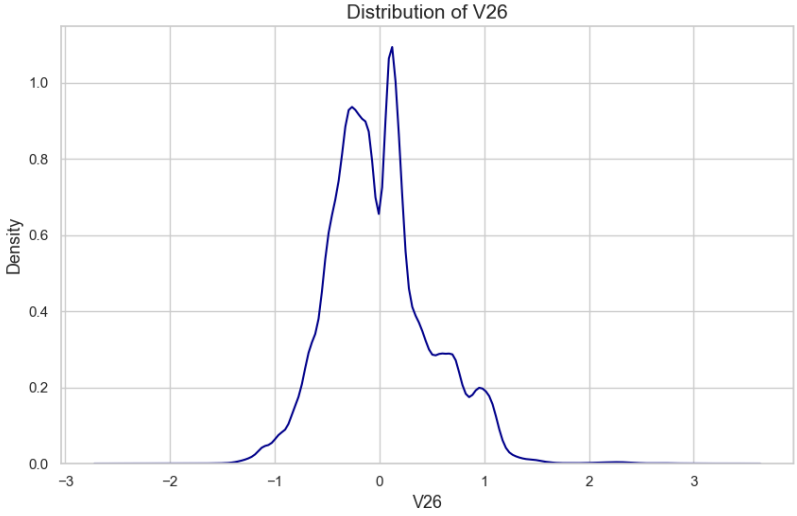

Fig. B.26: Density distribution of the feature V26.

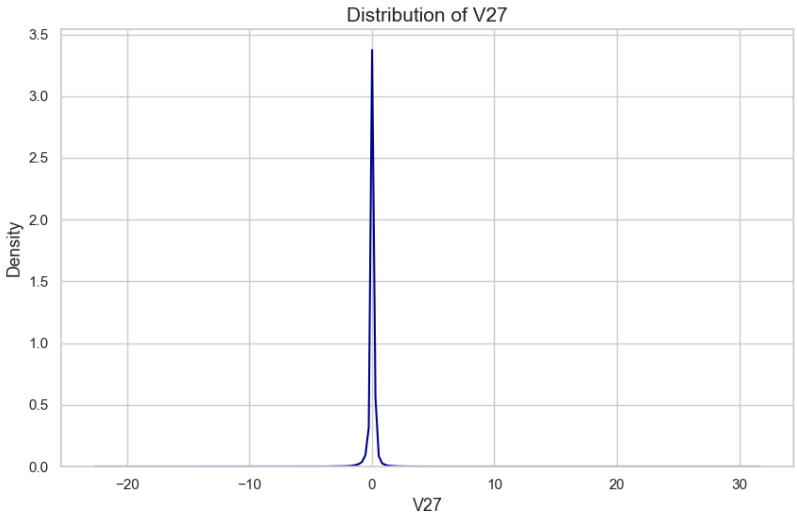

Fig. B.27: Density distribution of the feature V27.

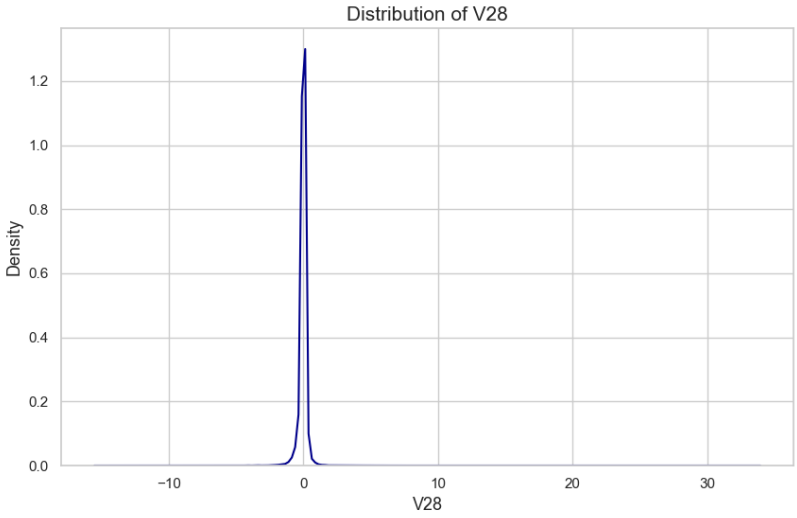

Fig. B.28: Density distribution of the feature V28.

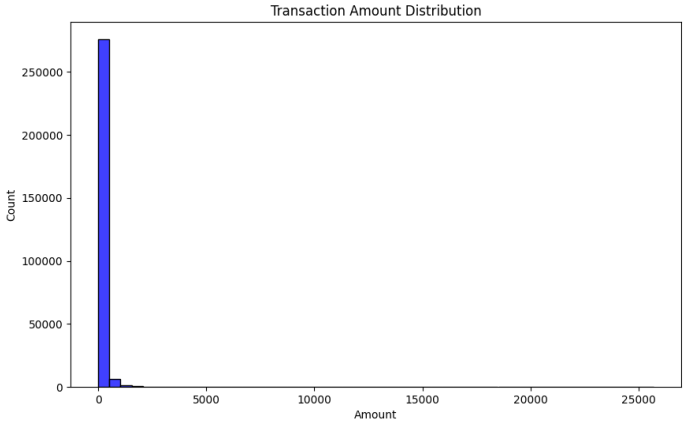

Fig. B.29: Distribution of Transaction Amounts

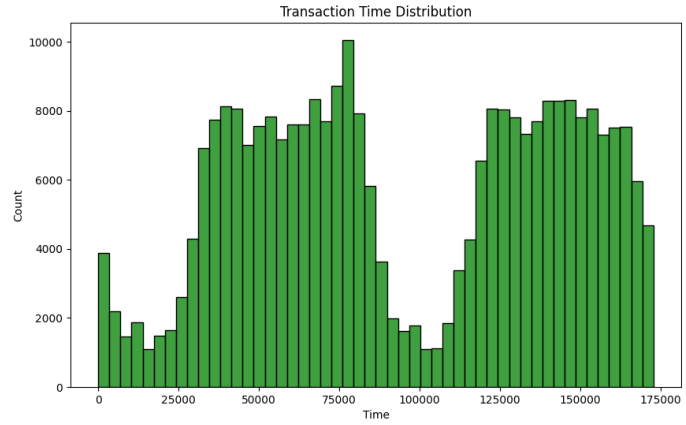

Fig. B.30: Distribution of Transaction Times

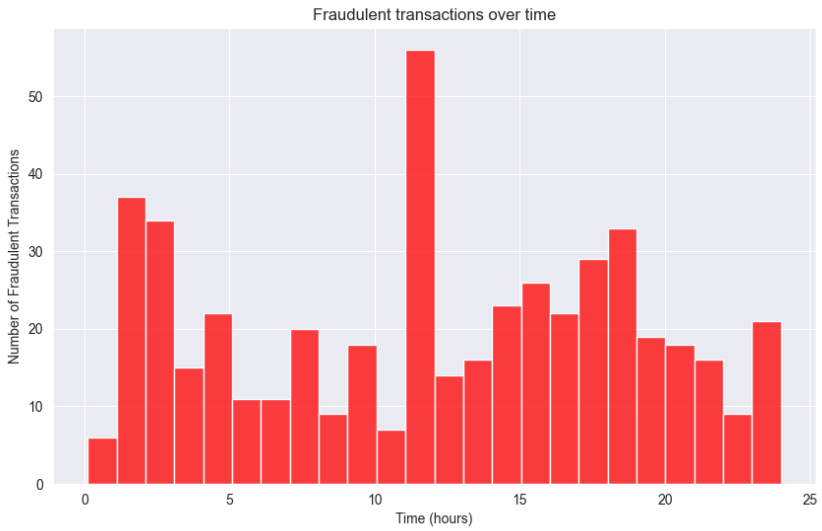

Fig. B.31: Histogram of fraudulent transactions over a 24-hour period

Appendix C. Results.

Table C.1: Results: precision, recall, F1-score, Average precision, AUPRC

	Precision	Recall	F1-score	Average precision	AUPRC
RF	**0.927273**	**0.864407**	**0.894737**	**0.907203**	**0.907172**
SDG_RF_0.01	0.909091	0.847458	0.877193	0.899719	0.899388
SDG_RF_0.05	0.942308	0.830508	0.882883	0.892200	0.891795
SDG_RF_0.1	0.924528	0.830508	0.875000	0.891340	0.890987
SDG_RF_0.25	0.924528	0.830508	0.875000	0.896192	0.895874
SDG_RF_0.5	**0.943396**	0.847458	**0.892857**	0.903272	**0.902989**
SDG_RF_0.75	0.924528	0.830508	0.875000	0.895849	0.895482
SDG_RF_1	**0.943396**	0.847458	**0.892857**	0.898441	**0.898245**

Table C.2: Comparative Statistics of Real and Synthetic Fraudulent Transactions

Feature	Real Mean	Synthetic Mean	Real Std	Synthetic Std
scaled_amount	1.284367	-0.105918	3.437048	0.353782
scaled_time	-0.065293	0.039108	0.569150	0.367101
V1	-4.318652	-0.052419	6.447295	0.348198
V2	3.385874	-0.058951	4.007597	0.298490
V3	-6.691748	-0.184022	6.832462	0.326876
V4	4.543605	-0.180708	2.803537	0.351087
V5	-2.824286	0.171967	5.171203	0.327737
V6	-1.451156	-0.066560	1.706642	0.252851
V7	-5.176784	-0.109007	6.777946	0.311330
V8	0.809420	-0.115566	5.622611	0.319582
V9	-2.514537	-0.016826	2.458720	0.323216
V10	-5.459115	0.099388	4.758916	0.414964
V11	3.759853	0.234665	2.722260	0.272847
V12	-6.211331	-0.071556	4.579088	0.423495
V13	-0.034655	-0.221552	1.093499	0.294859
V14	-6.942029	0.119838	4.309694	0.264631
V15	-0.072221	-0.071624	1.022020	0.336641
V16	-4.005238	0.089580	3.865921	0.345830
V17	-6.406285	-0.171757	6.972796	0.330061
V18	-2.126422	0.214821	2.895291	0.353486
V19	0.672789	-0.036218	1.602590	0.371307
V20	0.328545	-0.115584	1.196810	0.371320
V21	0.612619	0.230824	3.033279	0.347309
V22	0.042283	0.169966	1.236904	0.334313
V23	-0.120926	0.161449	1.624724	0.333287
V24	-0.106338	0.207110	0.516499	0.327641
V25	0.028429	0.157265	0.825721	0.346790
V26	0.081238	-0.026911	0.475545	0.330743
V27	0.183397	0.133662	1.329459	0.274445
V28	0.079234	0.135045	0.536673	0.349023

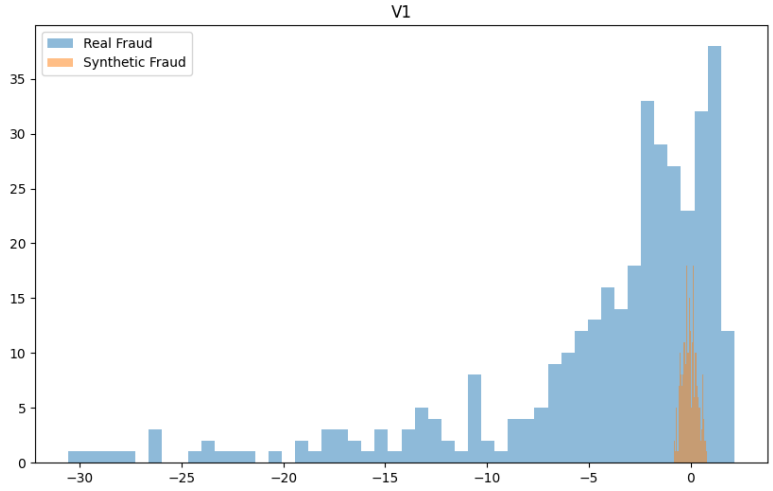

Fig. C.1: Real and Synthetic Data for V1

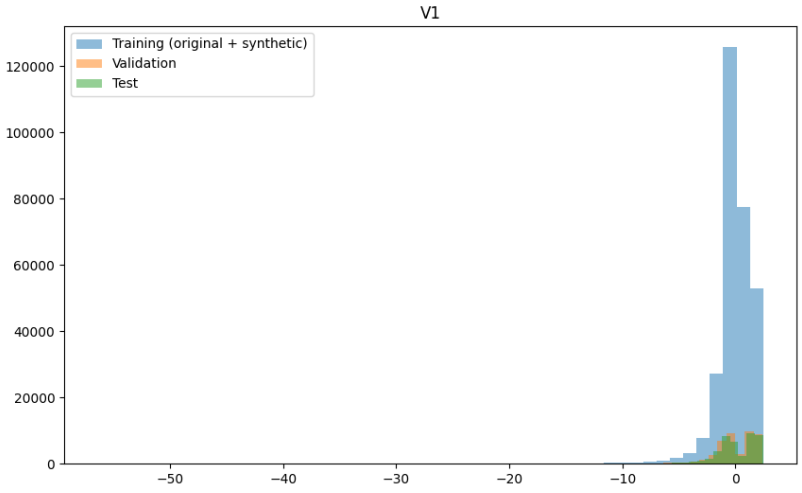

Fig. C.2: Sets Comparison for V1

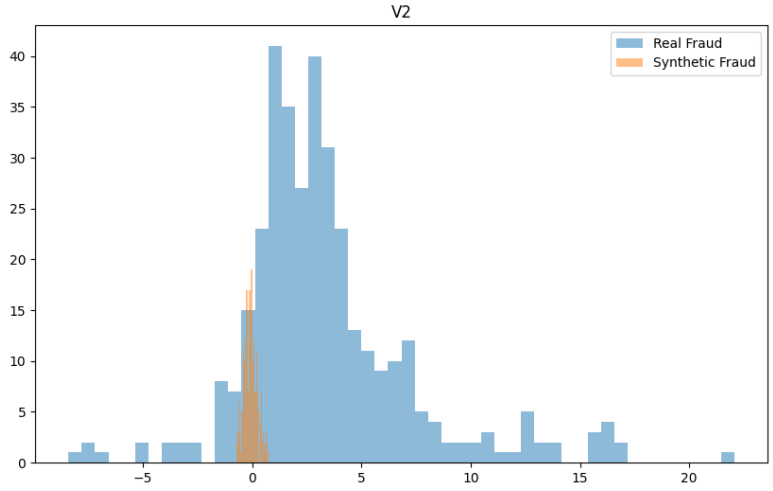

Fig. C.3: Real and Synthetic Data for V2

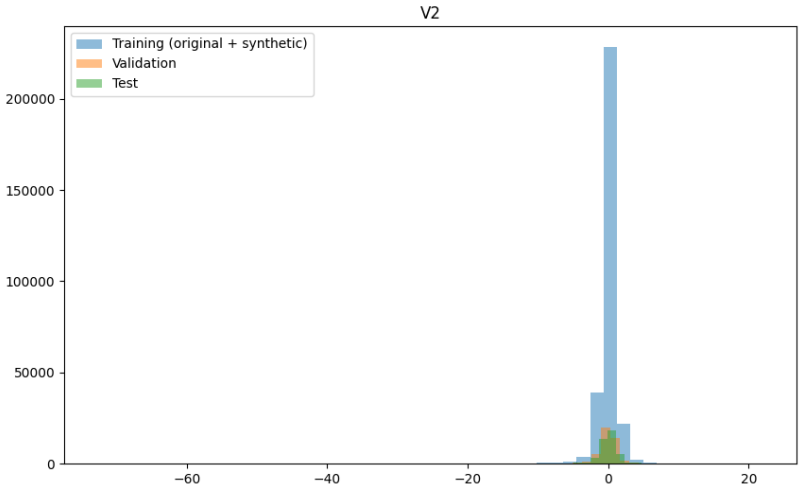

Fig. C.4: Sets Comparison for V2

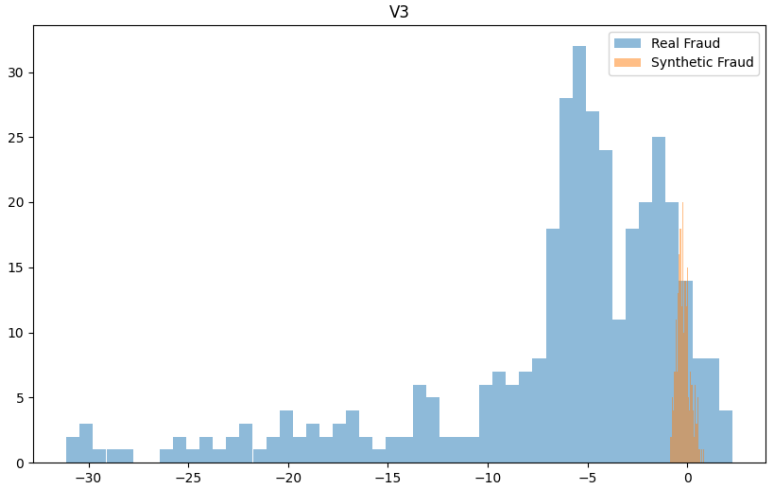

Fig. C.5: Real and Synthetic Data for V3

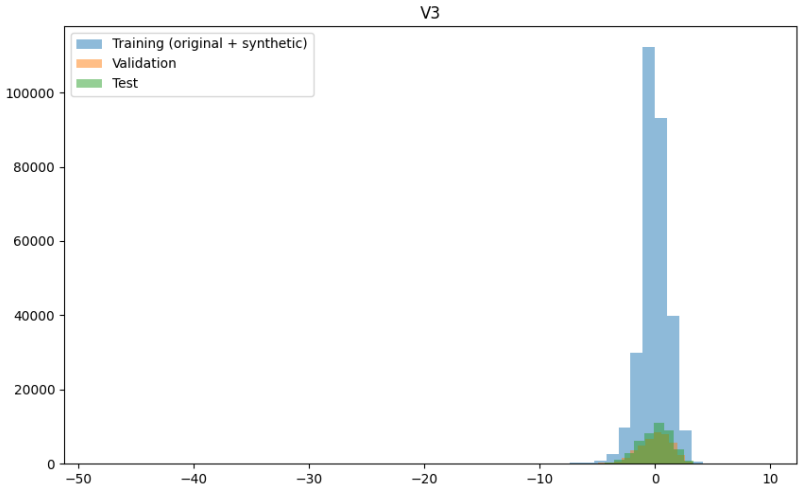

Fig. C.6: Sets Comparison for V3

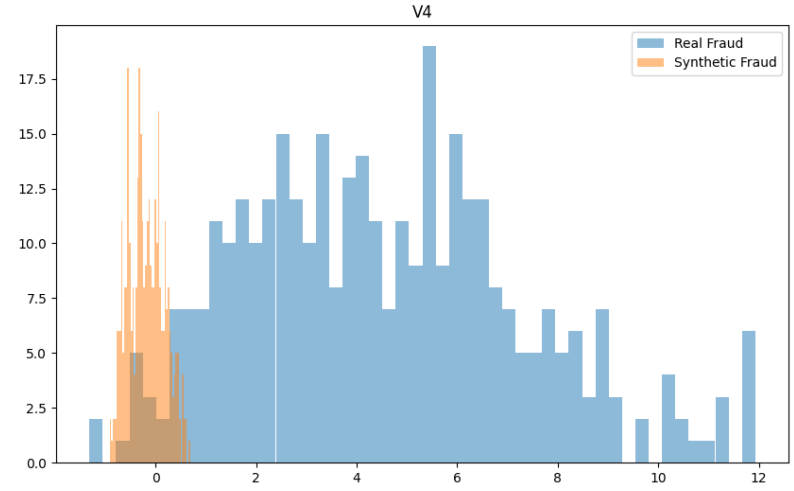

Fig. C.7: Real and Synthetic Data for V4

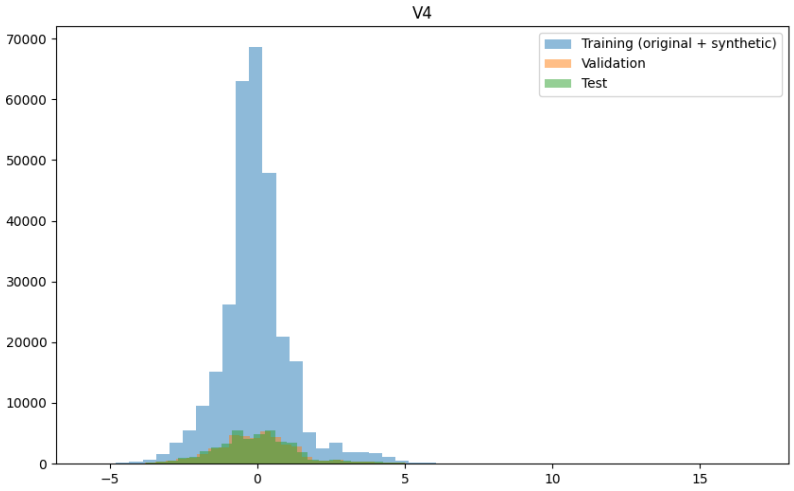

Fig. C.8: Sets Comparison for V4

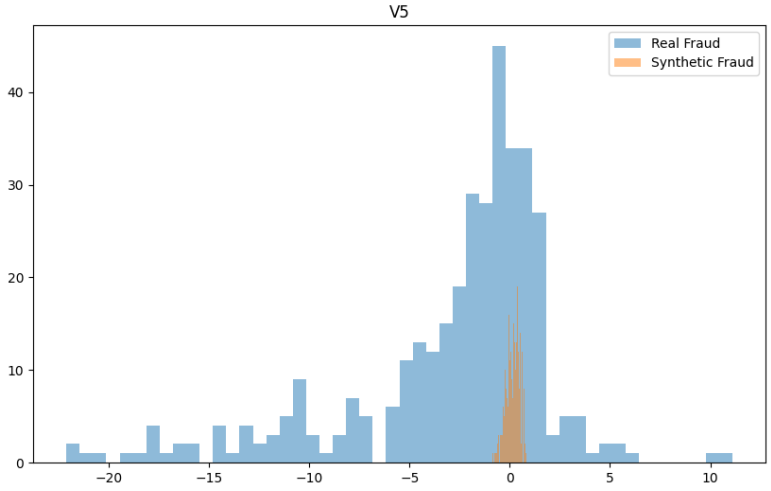

Fig. C.9: Real and Synthetic Data for V5

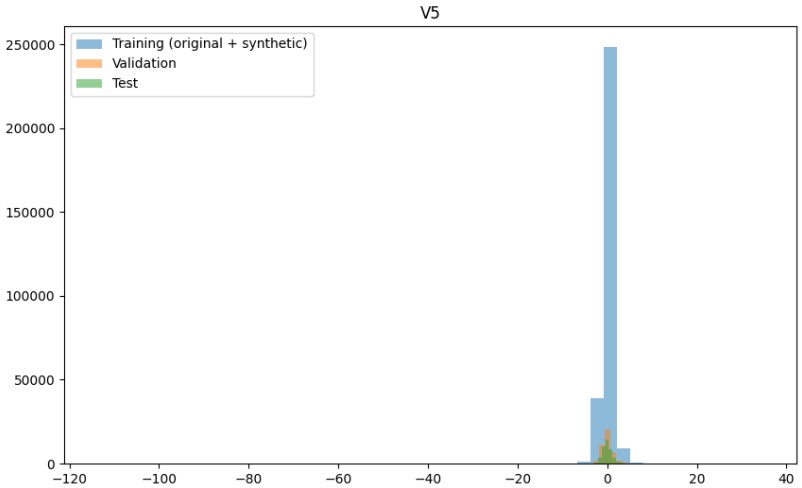

Fig. C.10: Sets Comparison for V5

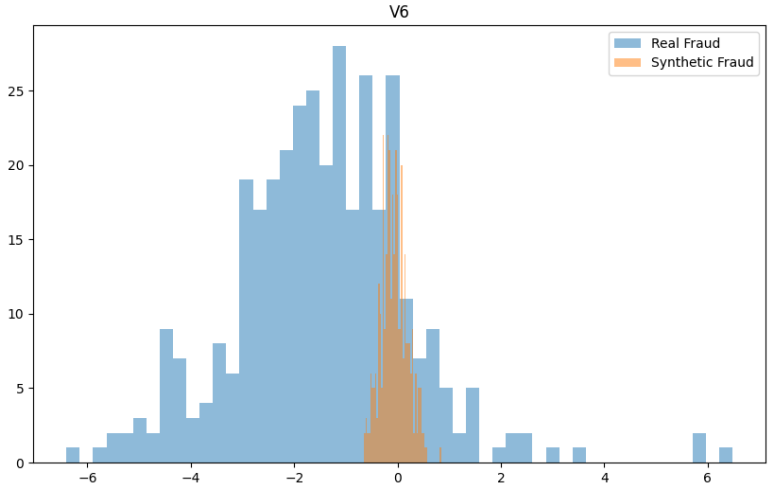

Fig. C.11: Real and Synthetic Data for V6

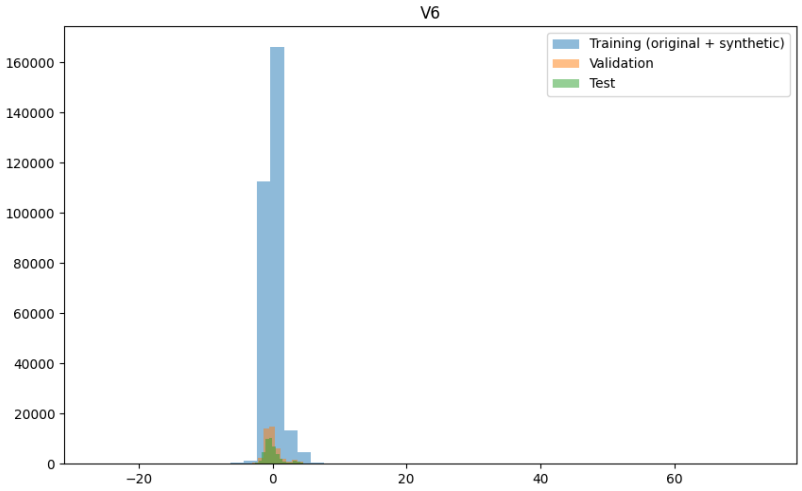

Fig. C.12: Sets Comparison for V6

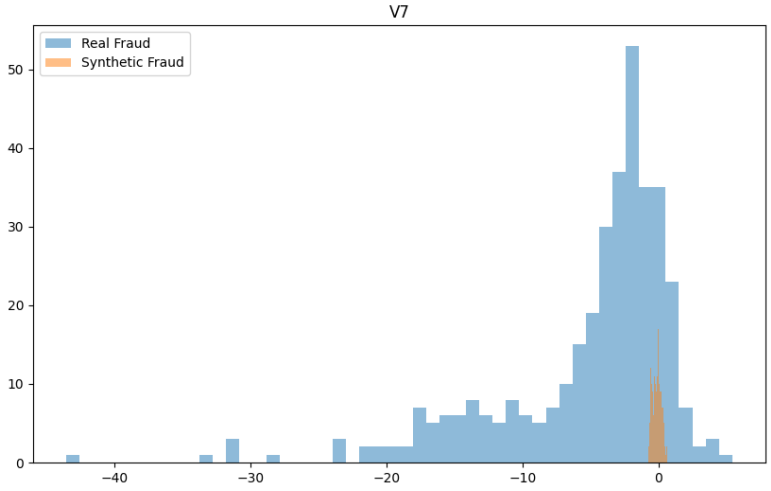

Fig. C.13: Real and Synthetic Data for V7

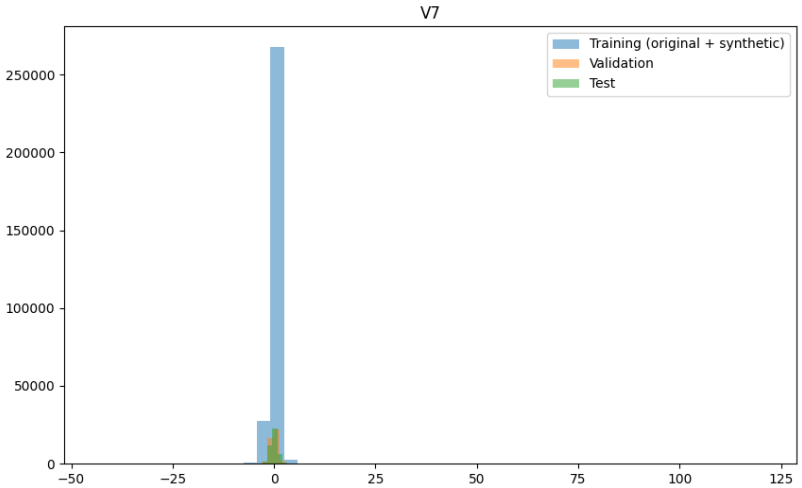

Fig. C.14: Sets Comparison for V7

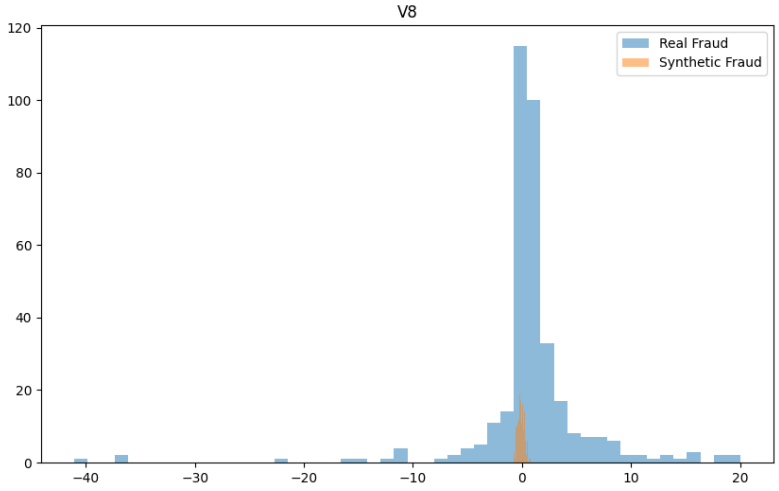

Fig. C.15: Real and Synthetic Data for V8

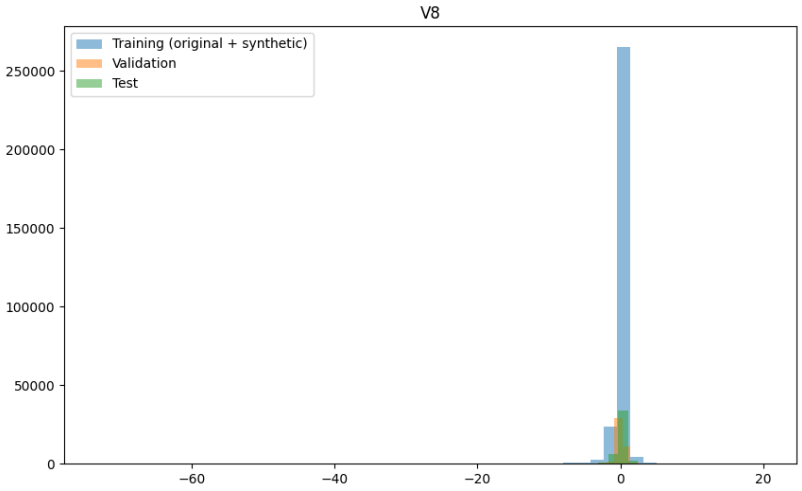

Fig. C.16: Sets Comparison for V8

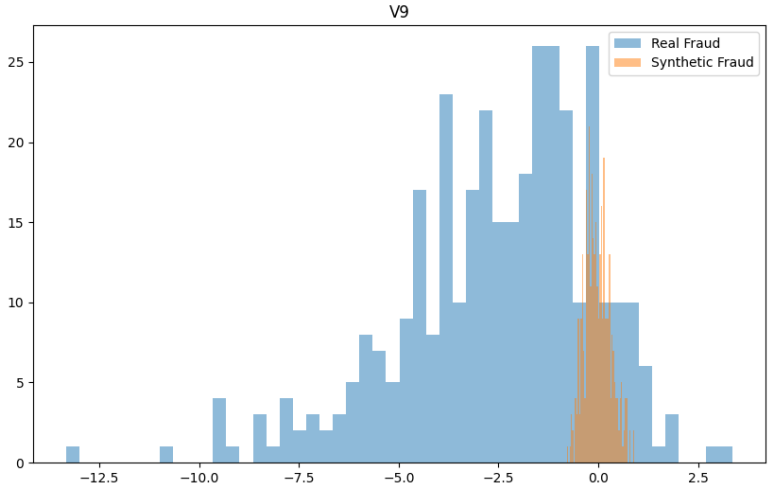

Fig. C.17: Real and Synthetic Data for V9

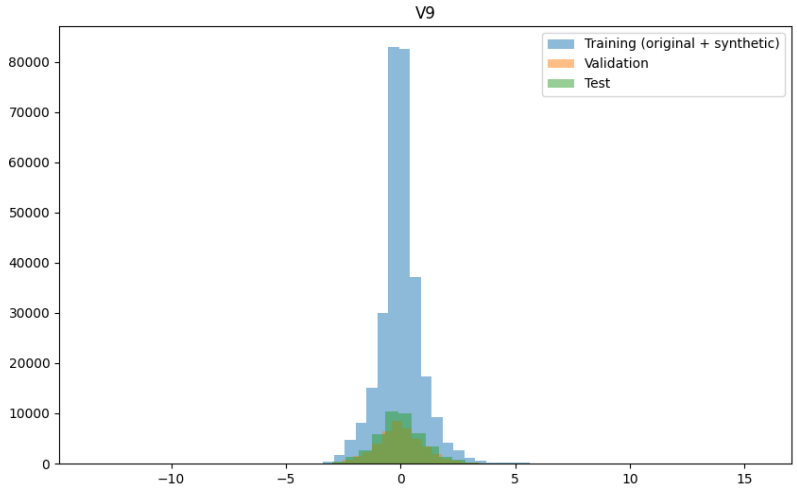

Fig. C.18: Sets Comparison for V9

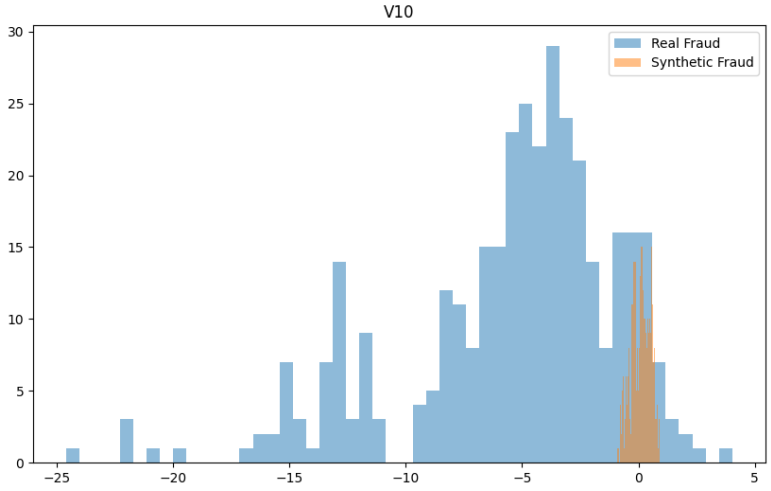

Fig. C.19: Real and Synthetic Data for V10

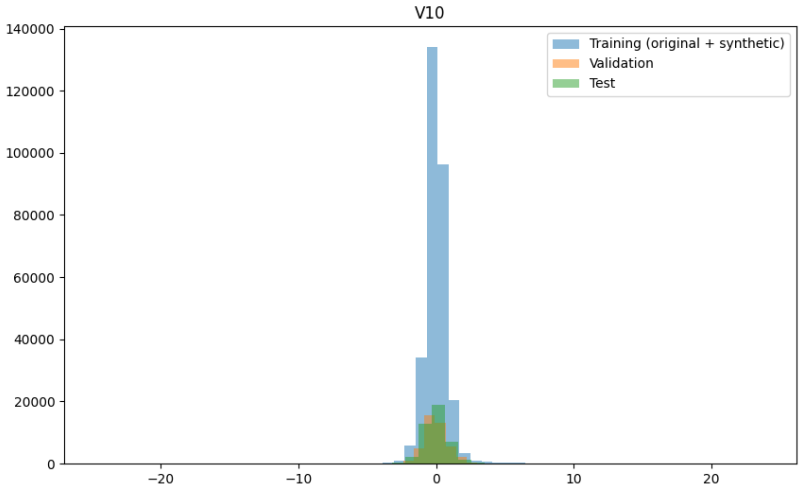

Fig. C.20: Sets Comparison for V10

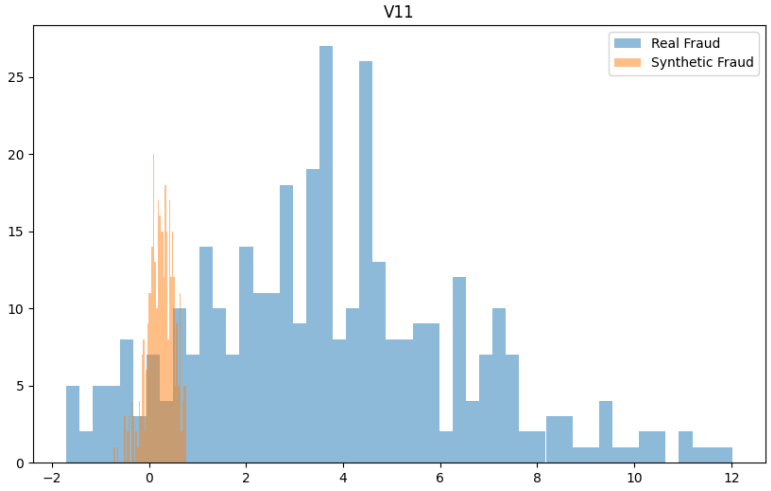

Fig. C.21: Real and Synthetic Data for V11

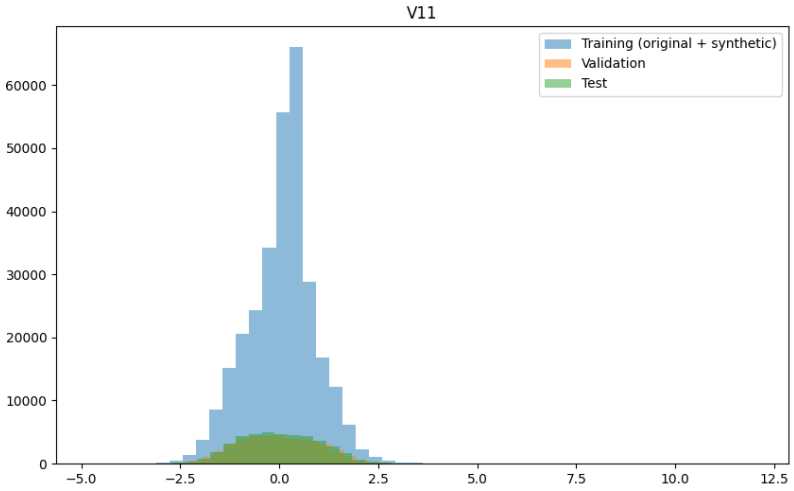

Fig. C.22: Sets Comparison for V11

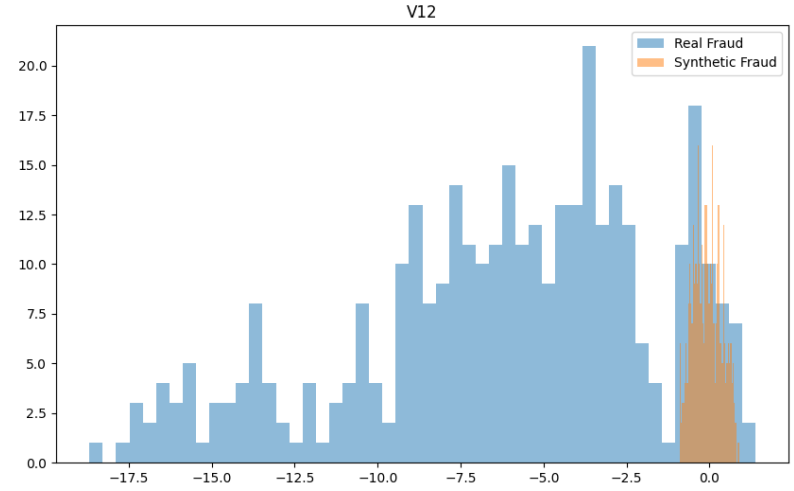

Fig. C.23: Real and Synthetic Data for V12

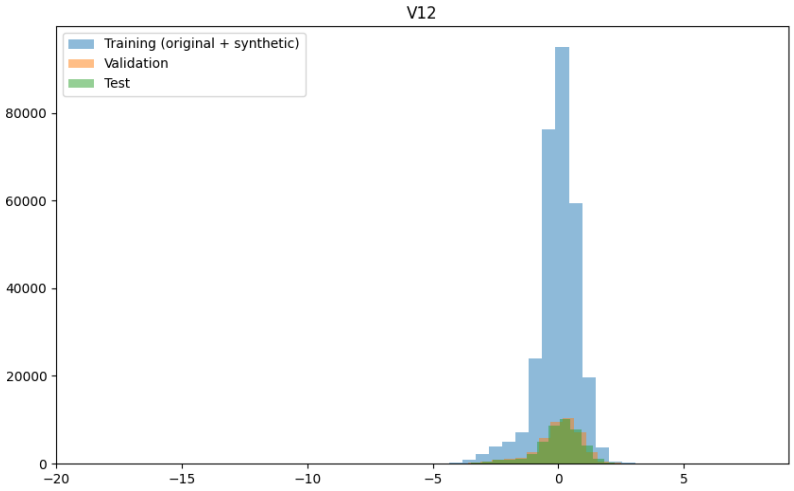

Fig. C.24: Sets Comparison for V12

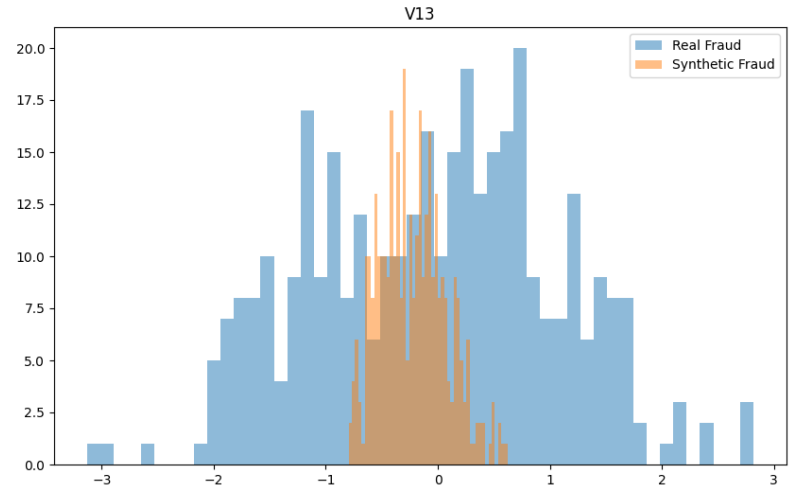

Fig. C.25: Real and Synthetic Data for V13

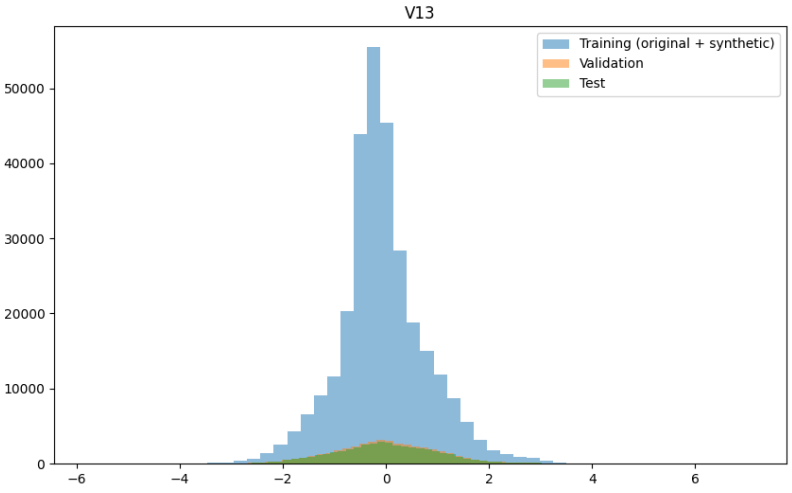

Fig. C.26: Sets Comparison for V13

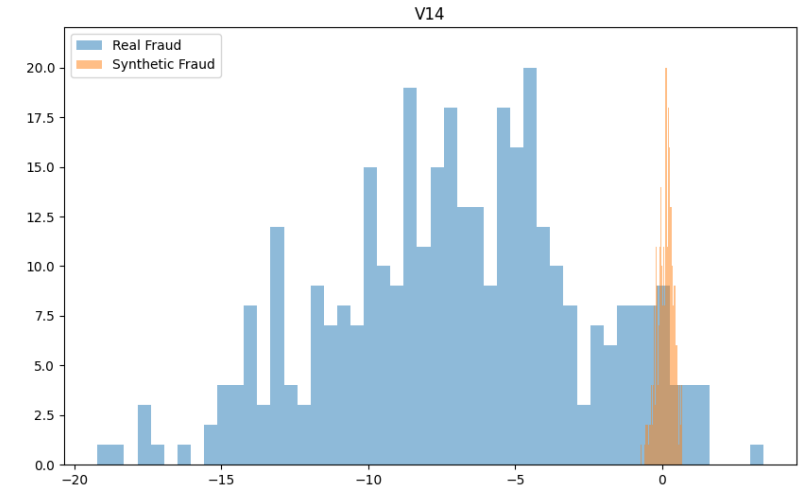

Fig. C.27: Real and Synthetic Data for V14

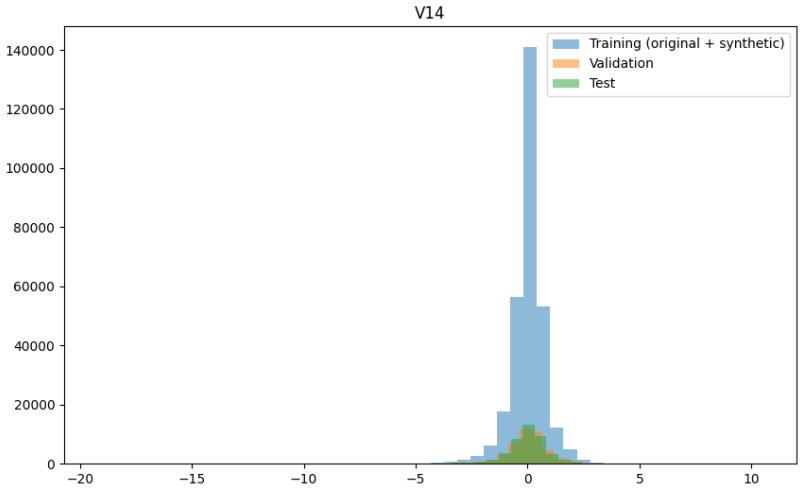

Fig. C.28: Sets Comparison for V14

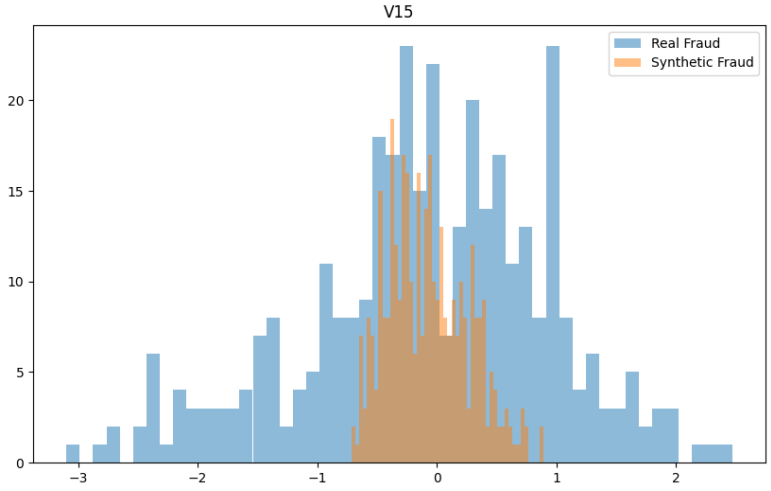

Fig. C.29: Real and Synthetic Data for V15

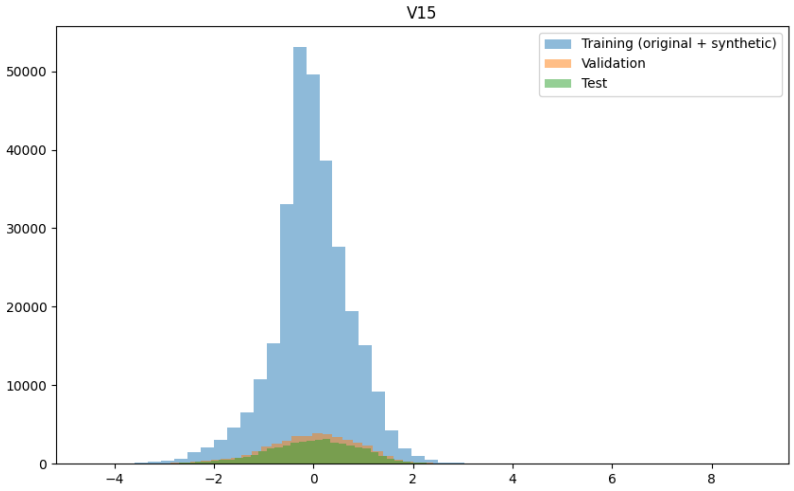

Fig. C.30: Sets Comparison for V15

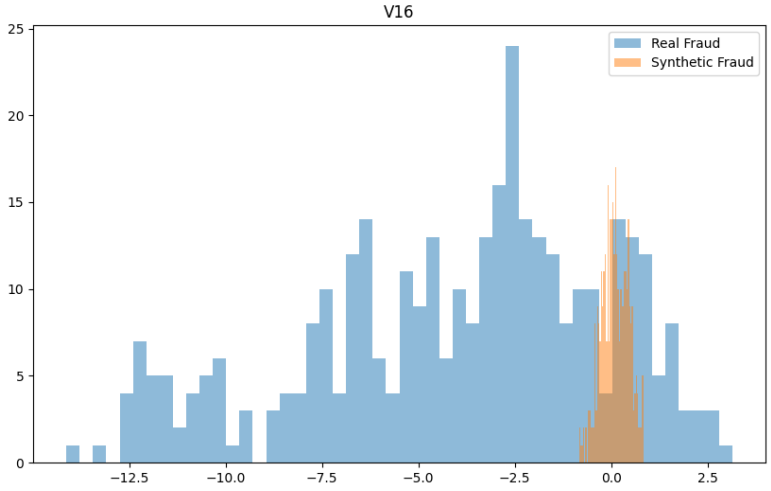

Fig. C.31: Real and Synthetic Data for V16

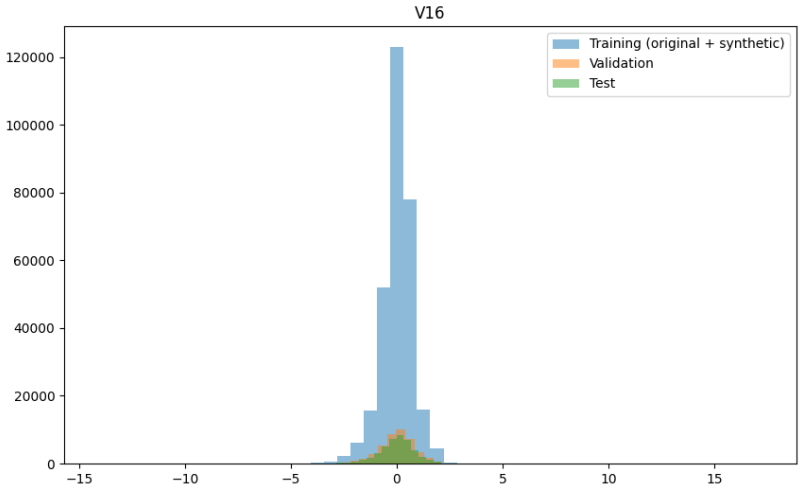

Fig. C.32: Sets Comparison for V16

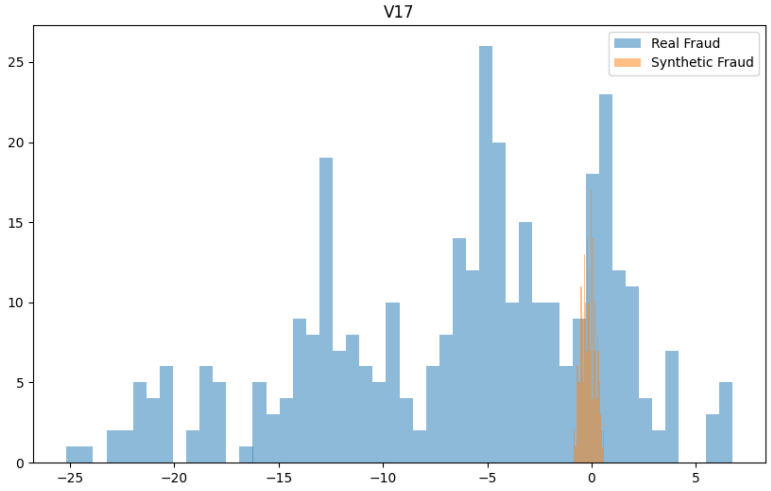

Fig. C.33: Real and Synthetic Data for V17

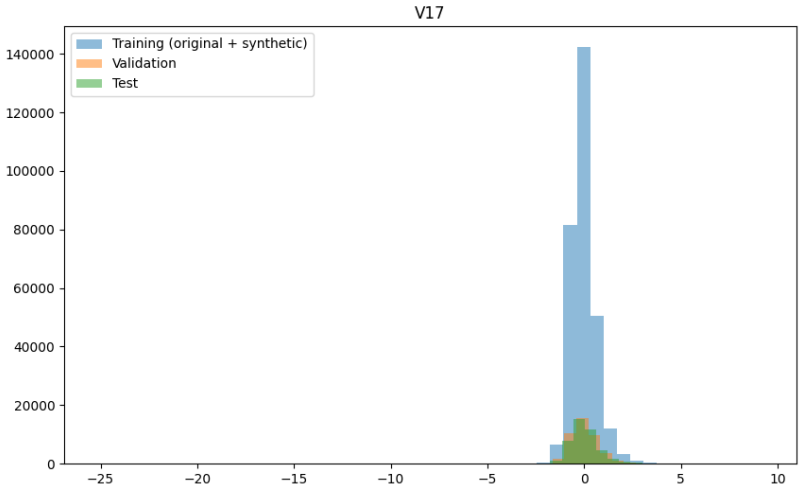

Fig. C.34: Sets Comparison for V17

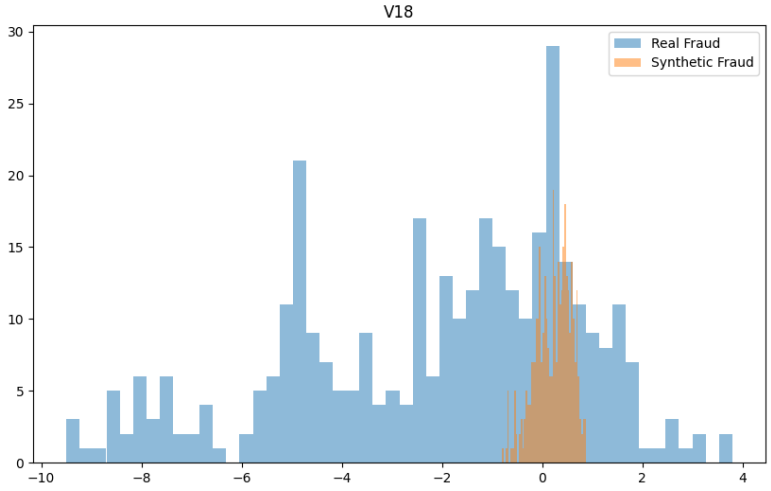

Fig. C.35: Real and Synthetic Data for V18

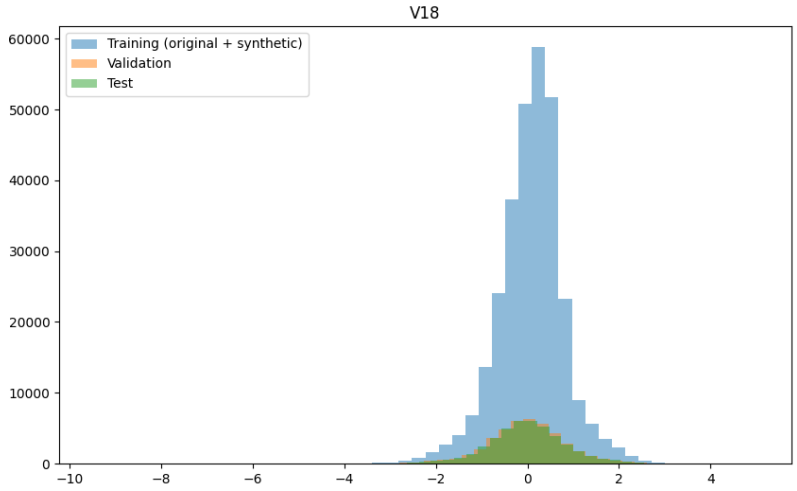

Fig. C.36: Sets Comparison for V18

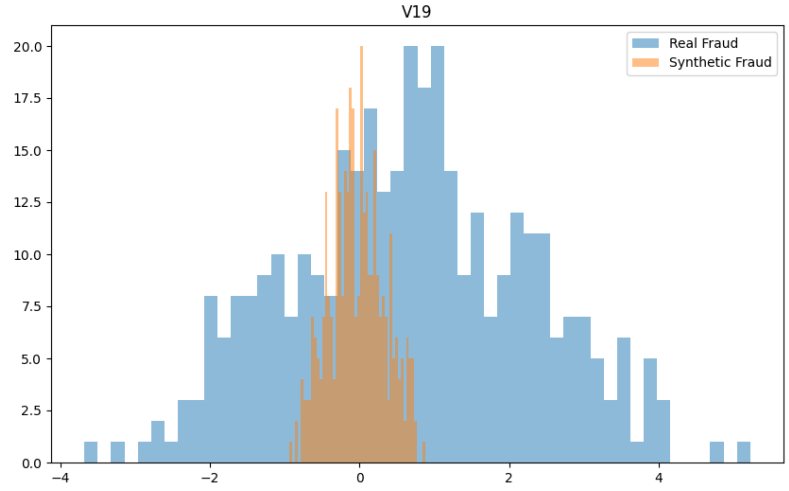

Fig. C.37: Real and Synthetic Data for V19

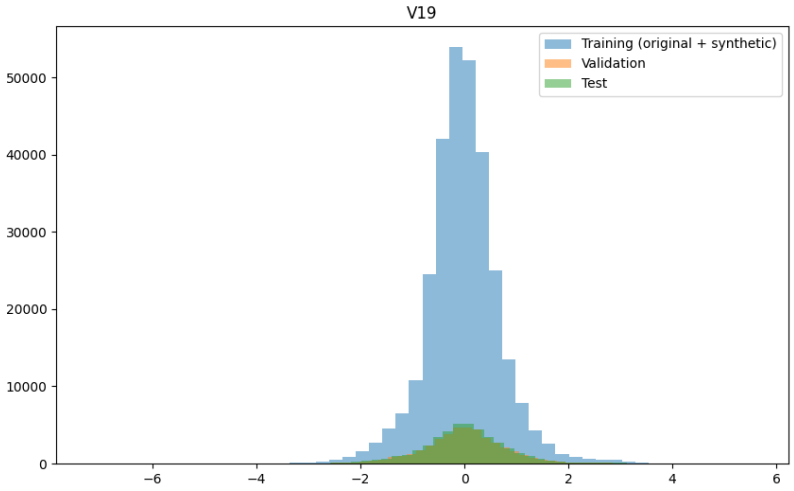

Fig. C.38: Sets Comparison for V19

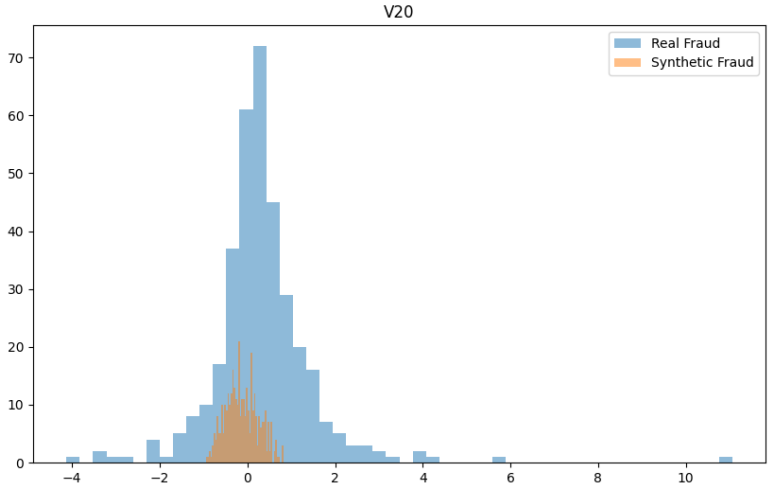

Fig. C.39: Real and Synthetic Data for V20

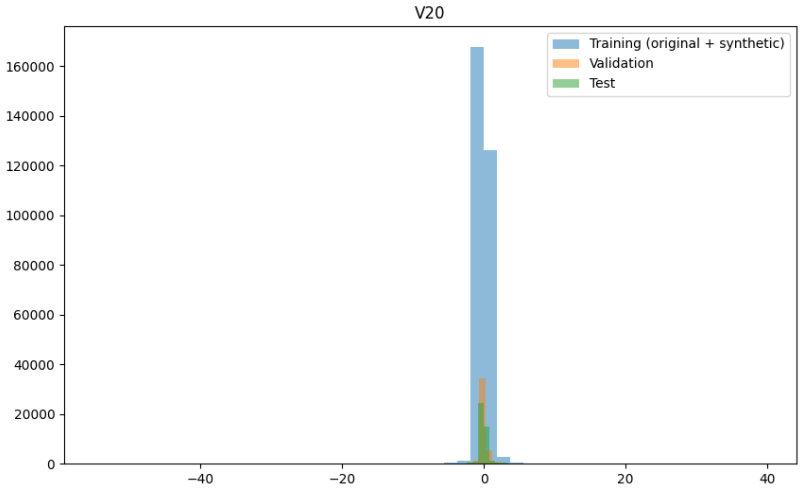

Fig. C.40: Sets Comparison for V20

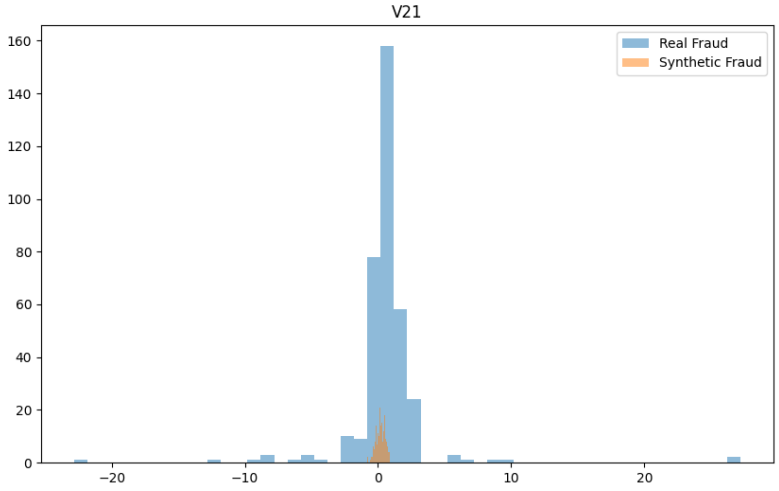

Fig. C.41: Real and Synthetic Data for V21

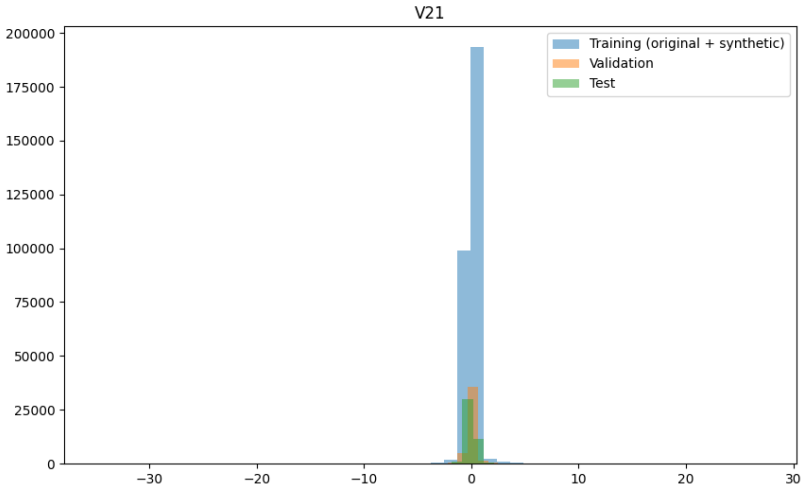

Fig. C.42: Sets Comparison for V21

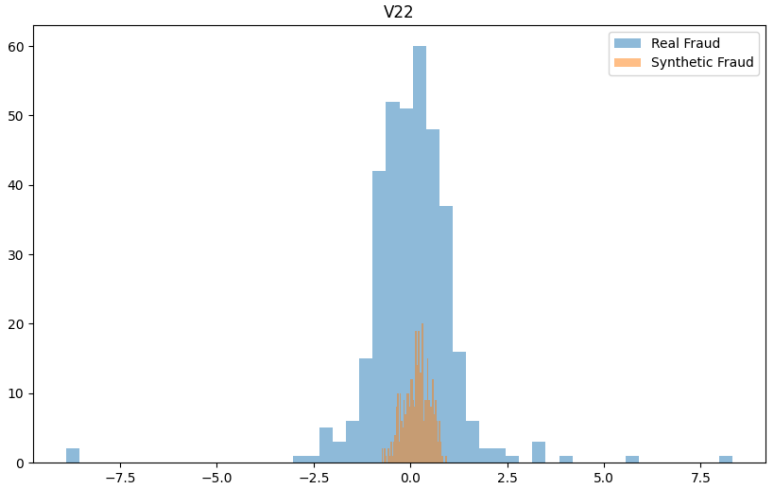

Fig. C.43: Real and Synthetic Data for V22

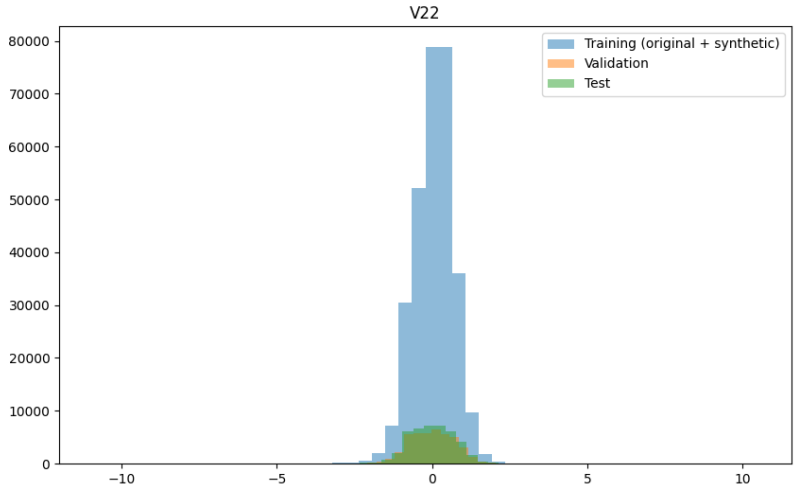

Fig. C.44: Sets Comparison for V22

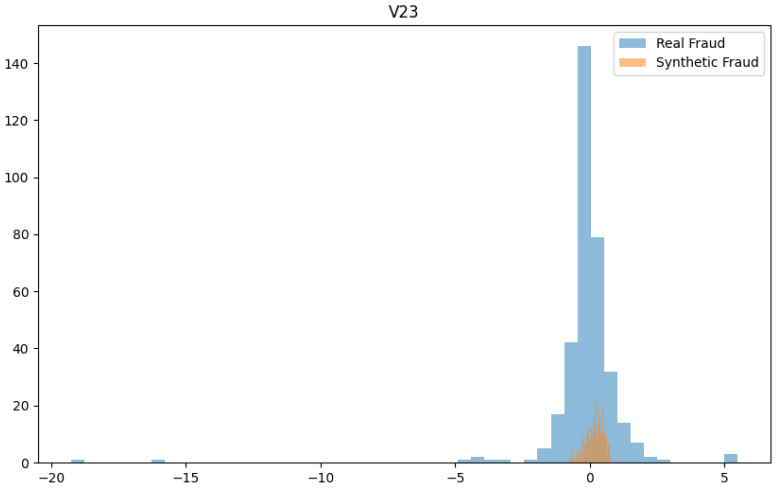

Fig. C.45: Real and Synthetic Data for V23

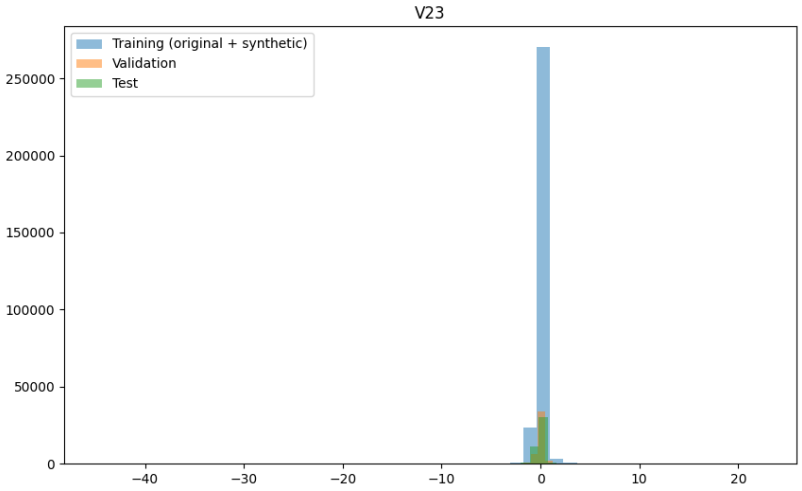

Fig. C.46: Sets Comparison for V23

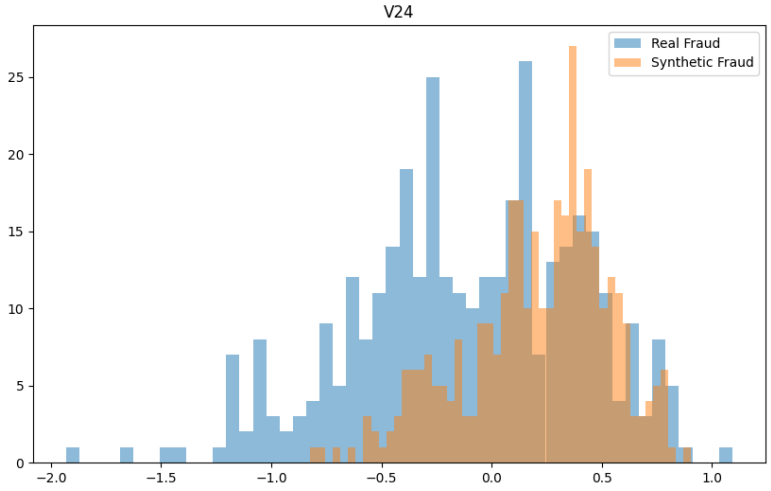

Fig. C.47: Real and Synthetic Data for V24

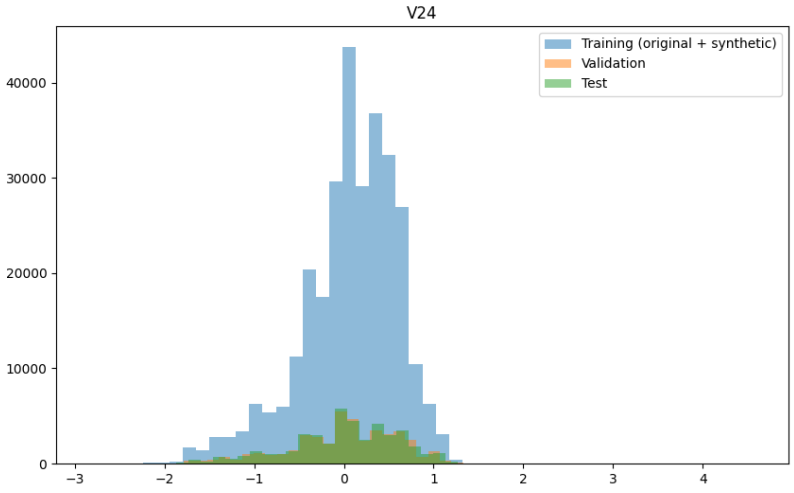

Fig. C.48: Sets Comparison for V24

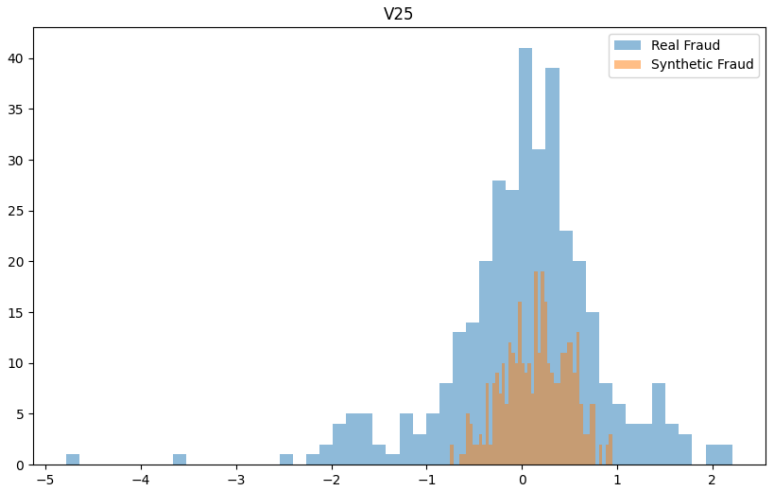

Fig. C.49: Real and Synthetic Data for V25

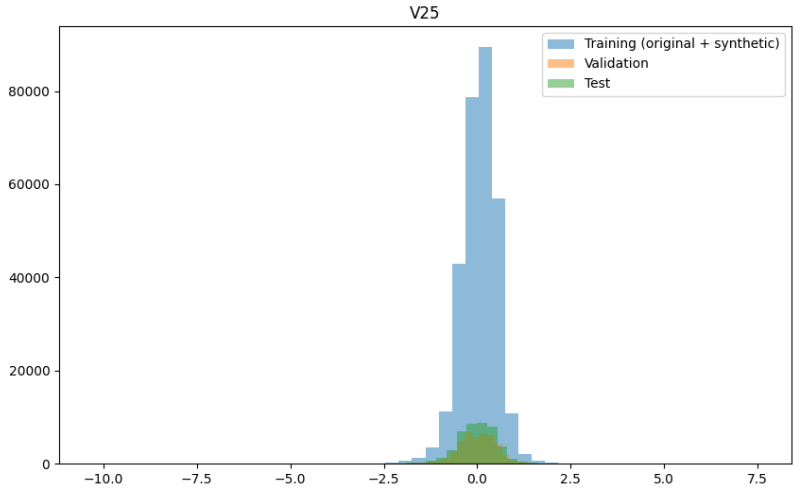

Fig. C.50: Sets Comparison for V25

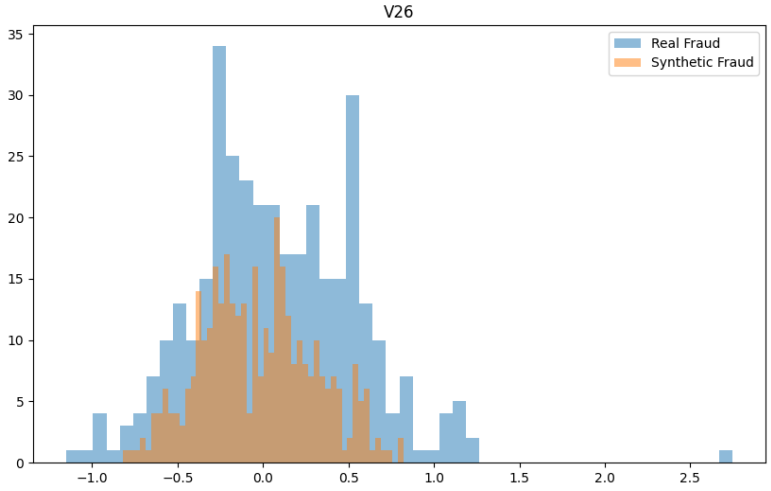

Fig. C.51: Real and Synthetic Data for V26

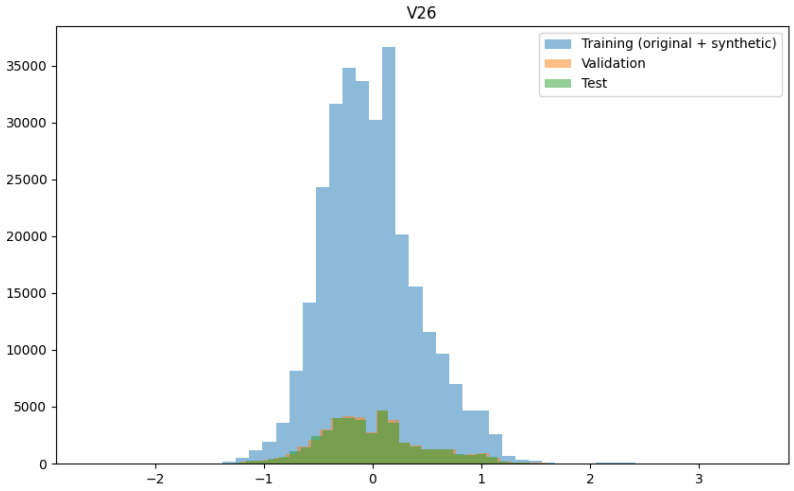

Fig. C.52: Sets Comparison for V26

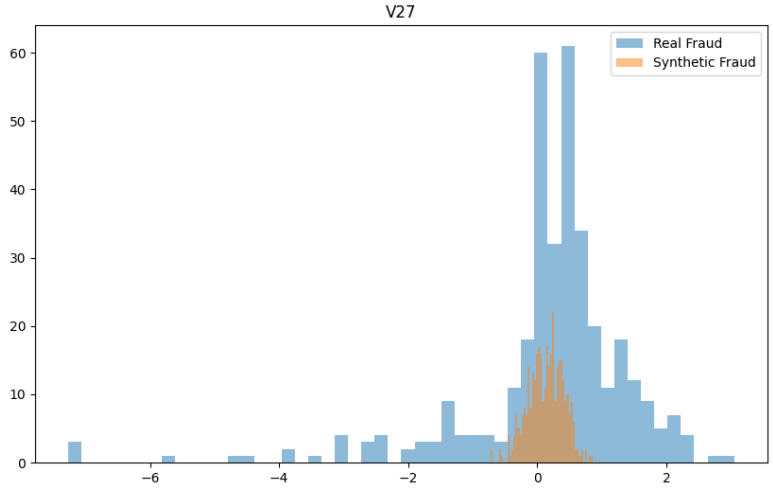

Fig. C.53: Real and Synthetic Data for V27

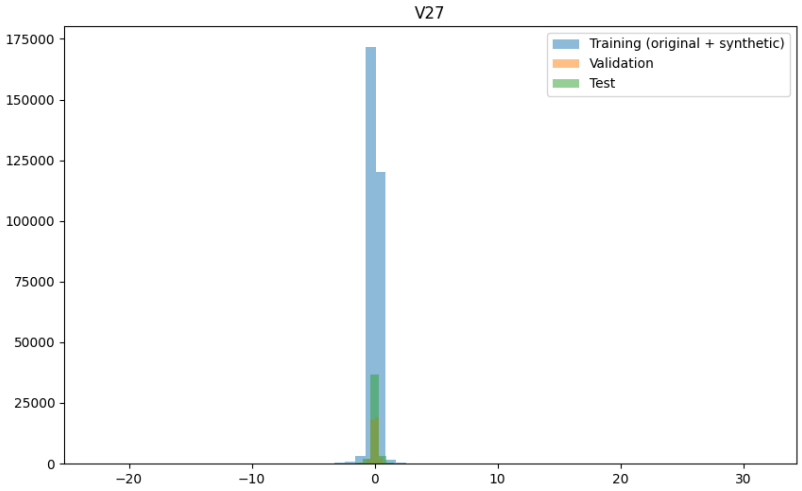

Fig. C.54: Sets Comparison for V27

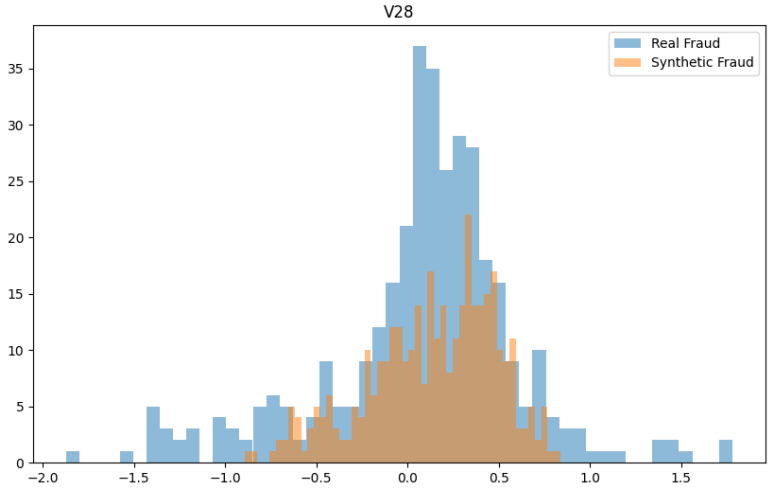

Fig. C.55: Real and Synthetic Data for V28

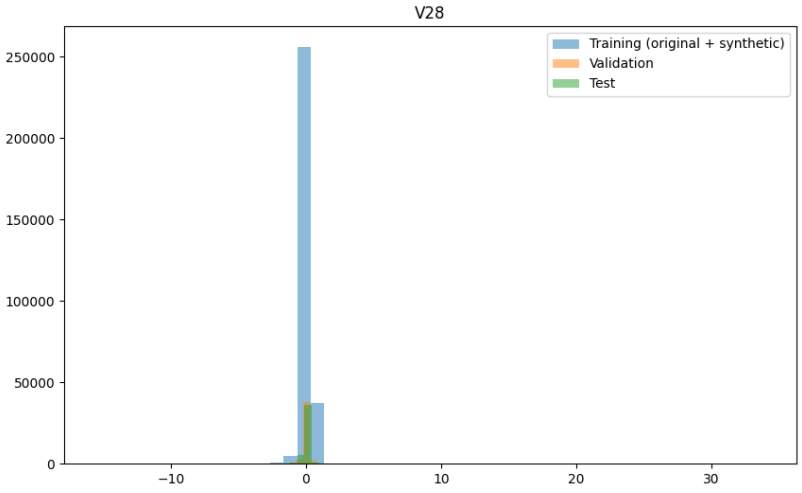

Fig. C.56: Sets Comparison for V28

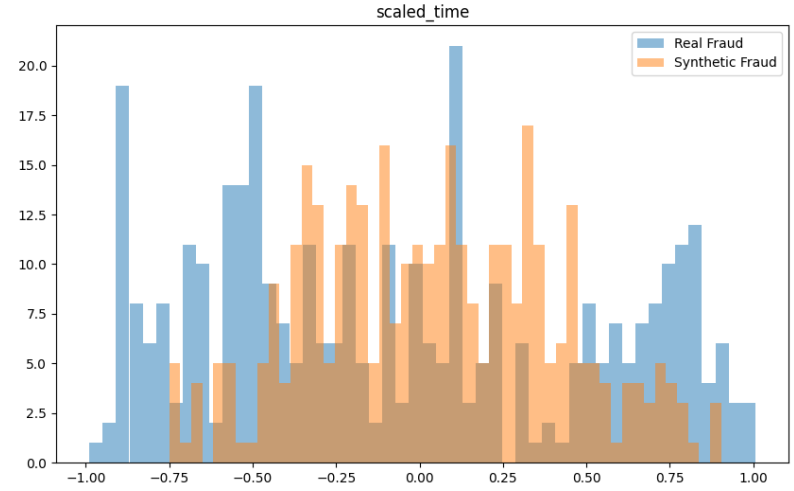

Fig. C.57: Real and Synthetic Data for scaled$_t ime$

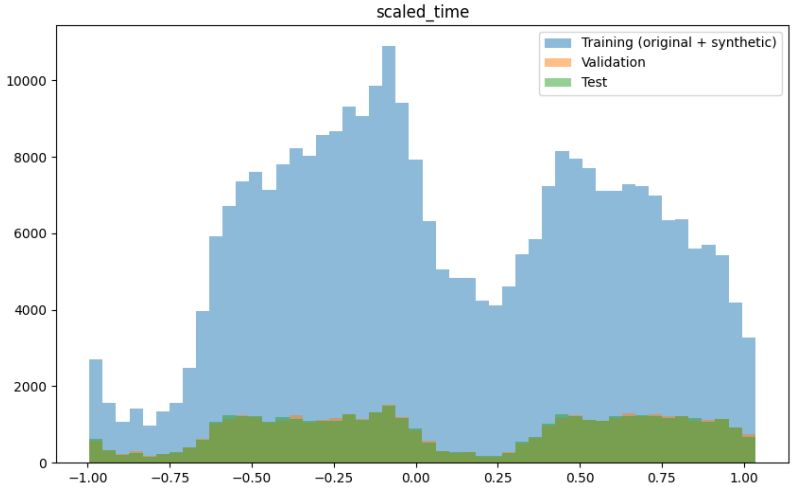

Fig. C.58: Sets Comparison for scaled$_t ime$

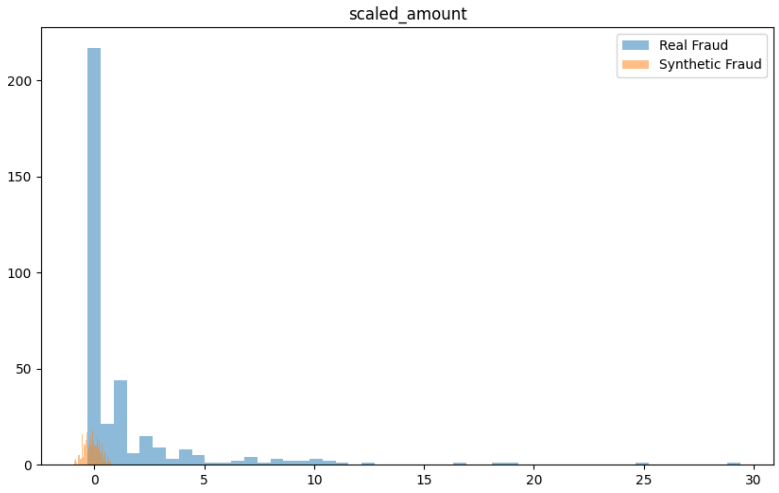

Fig. C.59: Real and Synthetic Data for scaled$_a$*mount*

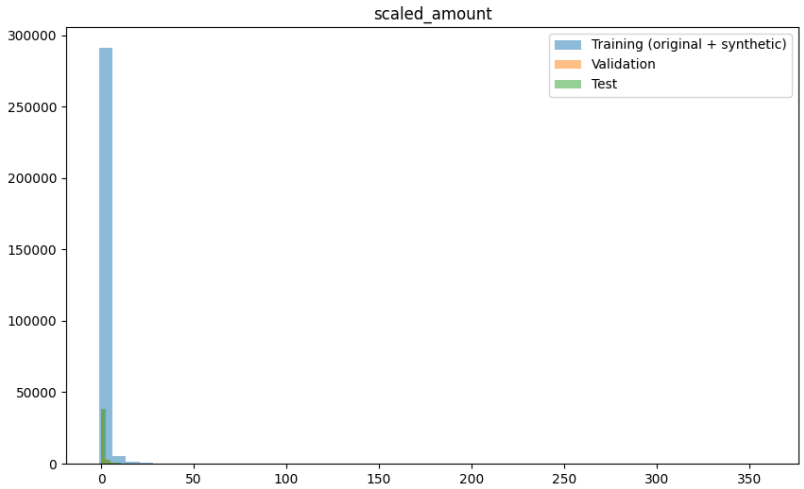

Fig. C.60: Sets Comparison for scaled$_a$*mount*

REFERENCES

[1] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y. *Generative Adversarial Nets*, in Advances in Neural Information Processing Systems, pages 2672–2680, 2014.

[2] Charitou, C., Dragicevic S., d'Avila Garcez A. *Synthetic Data Generation for Fraud Detection using GANs*, 2021.

[3] Langevin A., Cody T., Adams S., Beiling P. *Generative adversarial networks for data augmentation and transfer in credit card fraud detection*, in Journal of the Operational Research Society, 2021.

[4] Salimans T., Goodfellow I., Wojciech Z., Cheung V., Radford A., Chen X. *Improved Techniques for Training GANs*, in Advances in neural information processing systems, pages 2234-2242, 2016.

[5] Kaggle. *Credit Card Fraud Detection*. [online] Available at: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud [Accessed April 2023].