भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

**Indian Institute of Technology Hyderabad**

Fraud Analytics (CS6890)

Assignment: 5 | Synthetic data generation using Variational Autoencoder

| Name | Roll Number |
|---|---|
| Shreesh Gupta | CS23MTECH12009 |
| Hrishikesh Hemke | CS23MTECH14003 |
| Manan Patel | CS23MTECH14006 |
| Yug Patel | CS23MTECH14019 |
| Bhargav Patel | CS23MTECH11026 |

## Contents

## 1 | Problem statement

- Design and implement a Variational Autoencoder (VAE)-based synthetic data generation model to generate synthetic credit card transaction data. The goal is to create a model capable of generating 2 million synthetic transactions that closely resemble real credit card transactions from the provided dataset.

## 2 | Description of the data set

- **Size:** Each row represents a single transaction with corresponding attribute values. The card-transaction.v1.csv consist of 2,555,191 credit card transactions.
- **Fraudulent Transactions:** There are 3137 instances labeled as fraudulent. As majority of the transactions are non fraudulent it indicates that it's an unbalanced dataset
- **Attributes:** The dataset contains a total of 15 attributes:
  - User, card, year, month, day, time etc.
  - Also there is a column for "Error". This attribute indicates reason for transaction failure.
  - Last attribute "is Fraud?" (attribute value: Yes OR No)

## 3 | Our approach to solve the problem

### 3.1 | Data Exploration

- Firstly loading the **card-transaction.v1.csv** into the pandas dataframe. During data exploration there were many duplicates and Missing and Null values in the transactions and those attributes were **Amount, Use Chip, Merchant Name, Merchant City, Merchant State, Zip, MCC, Errors?, Is Fraud?** in the provided csv dataset file. Visualization the dataset in Box Plot as shown below.

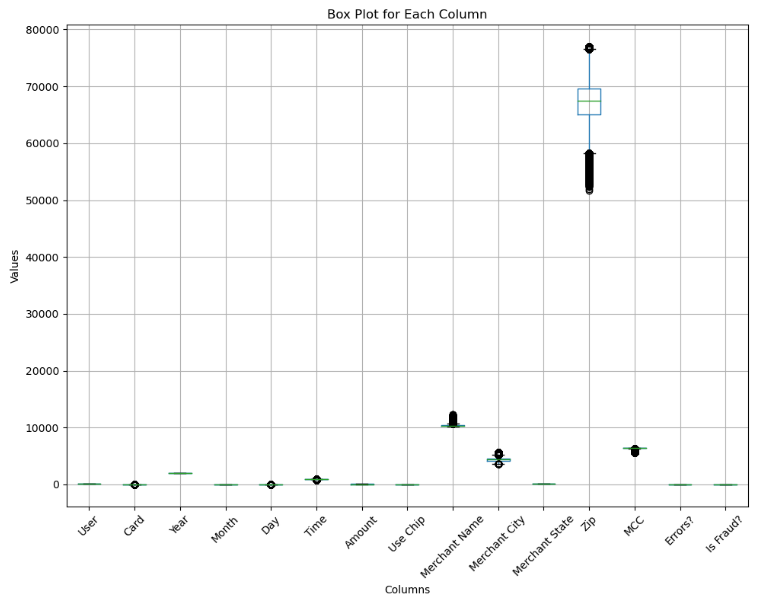


**Figure 3.1:** Box Plot for each attribute

- Finding relation between the missing values and the observed data. For both the below mentioned we found that there is no relation with Merchant City attribute. For both the plotted graph is same which is mentioned below.
  1. Merchant City VS Merchant State
  2. Merchant City VS Zip

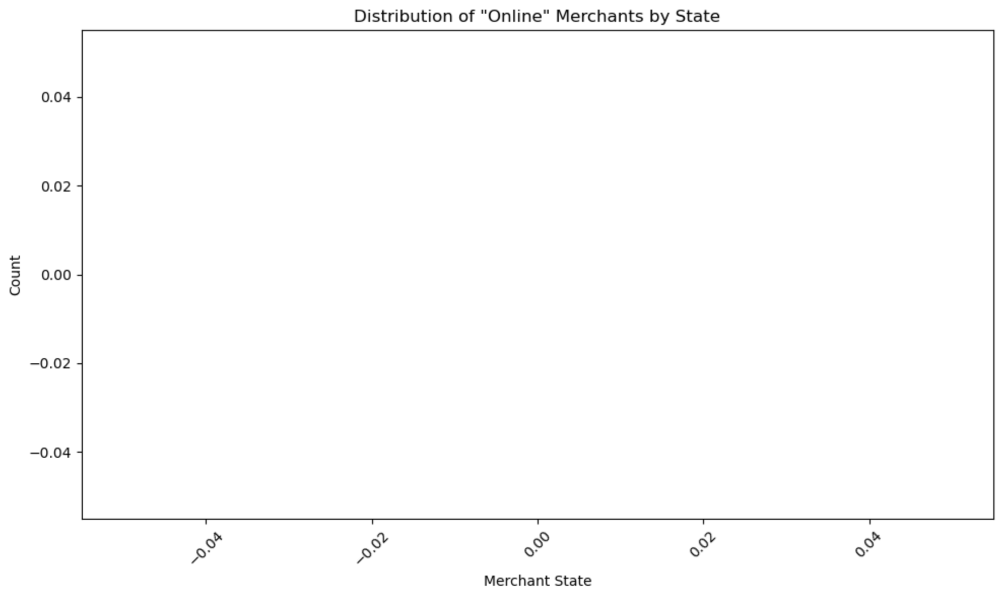


**Figure 3.2:** Distribution of ”Online” Merchant by State

- Now Filling the missing values. As these **Amount, Use Chip, Merchant Name, Merchant City, MCC, Is Fraud?** attributes where having only 1 missing value so we fill the missing value using mode (most frequent value) ensuring that the dataset remained consistent and preserve the categorical information effectively.
- Now the attributes which are having high frequency of missing values are **Merchant State, Zip, Error?**.
  - The missing values in the **Merchant State** column were filled with the placeholder **Unknown**.
  - For the **Zip** attribute, a custom approach was applied. Created the custom dictionary as Merchant City (key) and Zip (value) form already available data. Then filling the missing values in **Zip** attribute from the already created dictionary for Merchant City - Zip mapping. This method allowed for precise imputation of missing zip codes based on the known city information.
  - Missing values in the **Errors?** column were replaced with **No Error**, assuming that a missing entry indicated no reported errors.
- Also converting Time to minutes since midnight. Finally the dataset file **card-transaction.v1.csv** Exploratory Data Analysis is completed.

## 3.2 | Data Preprocessing

- The **Amount** attribute is converted to numeric format by removing the dollar sign.
- Categorical attributes **Use Chip, Merchant City, Merchant State, Errors?, Is Fraud?** are label encoded using scikit-learn's LabelEncoder.
- The **Merchant Name** attribute is converted from scientific notation to an integer format if necessary and then label encoded.
- Finally the preprocessed data is saved to a new CSV file named 'card-transaction-processed.csv'

## 3.3 | Varational Autoencoder for Finance-Tabular synthetic data generation

### 3.3.1 | Introduction to VAE

VAEs are a type of generative model that learns to encode and decode high-dimensional data in a latent space. Unlike traditional autoencoders, VAEs introduce a probabilistic approach, enabling them to capture the underlying distribution of the input data. This makes VAEs well-suited for generating new data samples that exhibit similar characteristics to the training data.

### 3.3.2 | Architecture

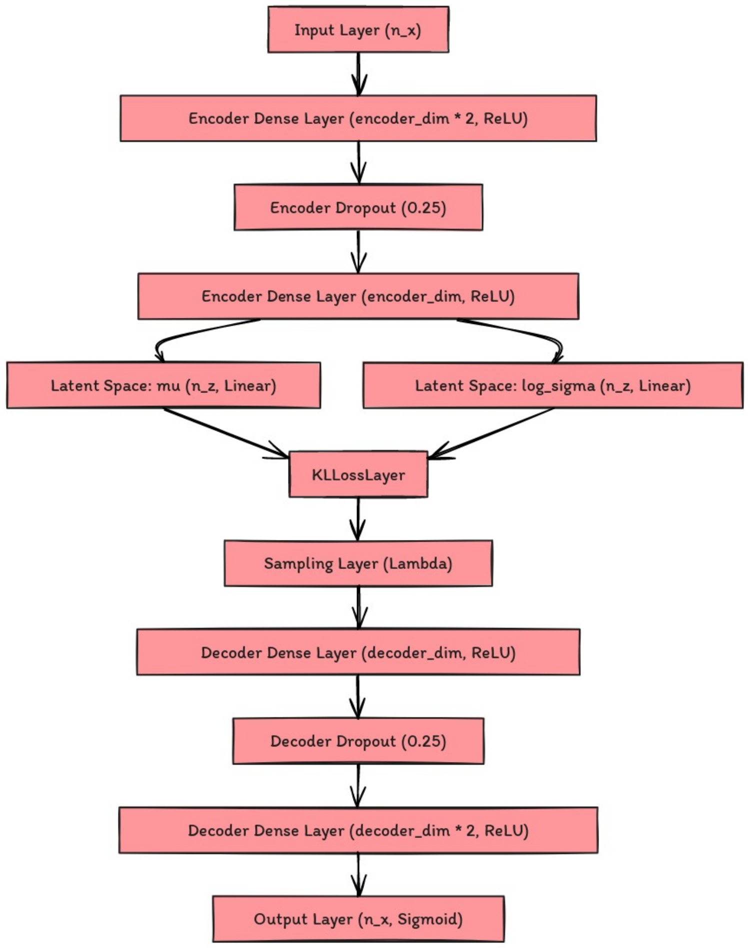

Input Layer (n_x)
Encoder Dense Layer (encoder_dim * 2, ReLU)
Encoder Dropout (0.25)
Encoder Dense Layer (encoder_dim, ReLU)
Latent Space: mu (n_z, Linear)
Latent Space: log_sigma (n_z, Linear)
KLLossLayer
Sampling Layer (Lambda)
Decoder Dense Layer (decoder_dim, ReLU)
Decoder Dropout (0.25)
Decoder Dense Layer (decoder_dim * 2, ReLU)
Output Layer (n_x, Sigmoid)

**Figure 3.3:** Architecture of the VAE that we have used

| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_3 (InputLayer) | (None, 15) | 0 |
| dense_7 (Dense) | (None, 10) | 160 |
| dropout_2 (Dropout) | (None, 10) | 0 |
| dense_8 (Dense) | (None, 5) | 55 |
| dense_9 (Dense) | (None, 2) | 12 |
| dense_10 (Dense) | (None, 2) | 12 |
| lambda_1 (Lambda) | (None, 2) | 0 |
| dense_11 (Dense) | (None, 5) | 15 |
| dropout_3 (Dropout) | (None, 5) | 0 |
| dense_12 (Dense) | (None, 10) | 60 |
| dense_13 (Dense) | (None, 15) | 165 |

| | |
|---|---|
| **Total params:** | 479 (1.87 KB) |
| **Trainable params:** | 479 (1.87 KB) |
| **Non-trainable params:** | 0 (0.00 Byte) |

- Here's an explanation of each layer and its purpose:

  1. **Input Layer (n-x)** This is where the input data is fed into the model. The notation n-x likely represents the dimensionality of the input vector.
  2. **Encoder Dense Layer (encoder-dim * 2, ReLU):** This is a fully connected (dense) layer that scales up the dimensionality to twice the encoder's dimensions. It uses ReLU (Rectified Linear Unit) as an activation function to add non-linearity to the model.
  3. **Encoder Dropout (0.25):** Dropout is used to prevent overfitting by randomly setting a fraction (25% here) of the input units to 0 at each update during training time.
  4. **Encoder Dense Layer (encoder-dim, ReLU):** 4. Another dense layer that brings the dimensionality down to the size of the encoder's dimension, again using ReLU for non-linearity.
  5. **Latent Space: mu (n-z, Linear) Latent Space: log-sigma (n-z, Linear):** These two layers output the parameters of the variational aspect of the VAE, mu and log-sigma, which define the mean and log variance of the Gaussian distribution from which the latent variables are sampled. n-z is the dimensionality of the latent space, and Linear activation is used to obtain actual values without transformation.
  6. **KLLossLayer:** This is a custom layer that calculates the Kullback-Leibler divergence loss, which measures how much information is lost when using the distribution defined by mu and log-sigma to represent the data compared to the prior distribution (usually a standard Gaussian).
  7. **Sampling Layer (Lambda):** This layer samples from the latent space using the mu and log-sigma to create a latent vector. The Lambda designation suggests the use of a custom function for sampling, typically involving the reparameterization trick for backpropagation.
  8. **Decoder Dense Layer (decoder-dim, ReLU):** This is the first layer of the decoder, which maps the latent vector back to the original space. It uses a ReLU activation function.
  9. **Decoder Dropout (0.25):** Similar to the encoder dropout, this layer is used in the decoder to prevent overfitting.
  10. **Decoder Dense Layer (decoder-dim * 2, ReLU):** This dense layer further scales up the dimensions to twice the decoder's dimension size with ReLU activation.
  11. **Output Layer (n-x, Sigmoid):** The final layer of the VAE which outputs the reconstruction of the input data. It has the same dimensionality as the input layer and uses a sigmoid activation function to ensure the output values are in the range [0,1], which is particularly useful if the input data has been normalized to this range.

  - The Kullback-Leibler divergence loss ensures that the latent space is continuous and allows for effective generation of new data.

### 3.3.3 | The CRUX (custom layers for KL loss computation)

- Variational Autoencoder (VAE) with custom layers for KL loss computation, utilizing a more complex architecture with dropout regularization and advanced decoder layers. The VAE loss combines reconstruction loss with the KL divergence added by the custom layer, and the model is compiled with an Adam optimizer and specified learning rate for training.
- **Objective Function:**

$$\text{VAE Loss} = \text{Reconstruction Loss} + \text{KL Divergence Loss}$$

$$\text{KL Divergence Loss} = -0.5 \sum_{j=1}^{n_z} (1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2)$$

$$\text{Reconstruction Loss} = \frac{1}{N} \sum_{i=1}^{N} (y_{\text{true}}^{(i)} - y_{\text{pred}}^{(i)})^2$$

### 3.3.4 | Training results

- Loss vs epoch graph:

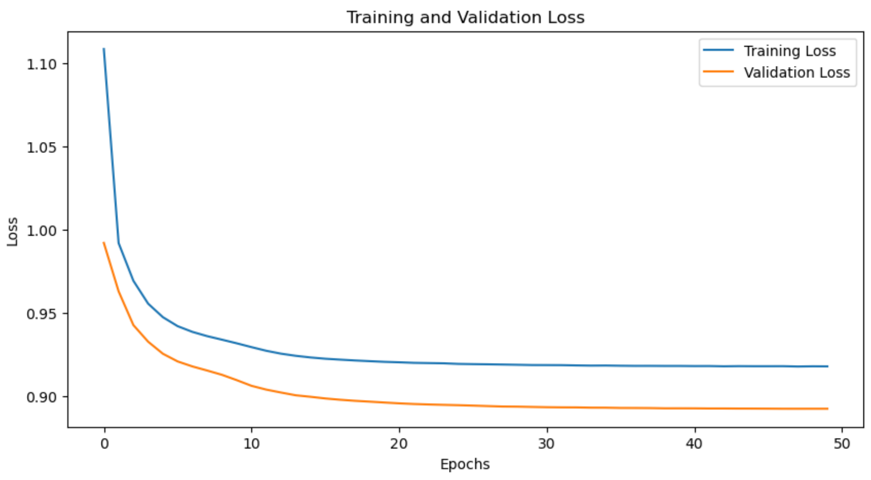


**Figure 3.4:** Training and Validation loss

### 3.3.5 | Encoder Model Creation and Data Transformation

- An encoder model using layers from an existing Variational Autoencoder (VAE) model. It extracts the latent representations of the input data by passing it through the encoder layers. The encoder-model is then used to transform the training data into the latent space, which is crucial for tasks like dimensionality reduction or feature extraction.

### 3.3.6 | Latent Space Visualization

- Scatter plot to visualize the latent representations of data points in a 2D latent space. Using same seed as used earlier for train/valid split. This visualization helps for interpreting the learned representations in a VAE.

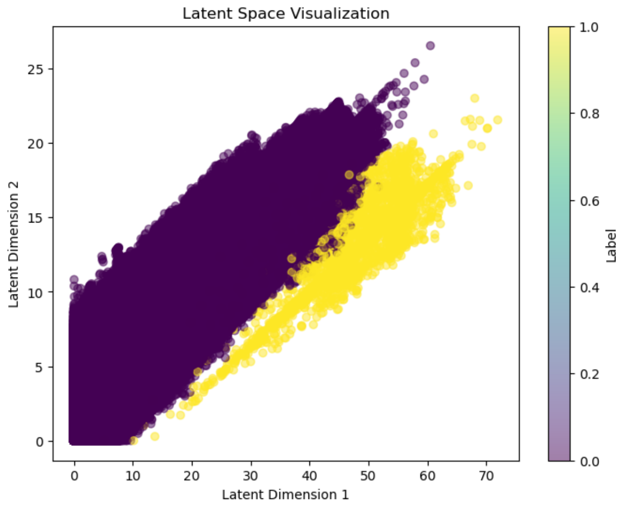


**Figure 3.5:** Latent Space Visualization

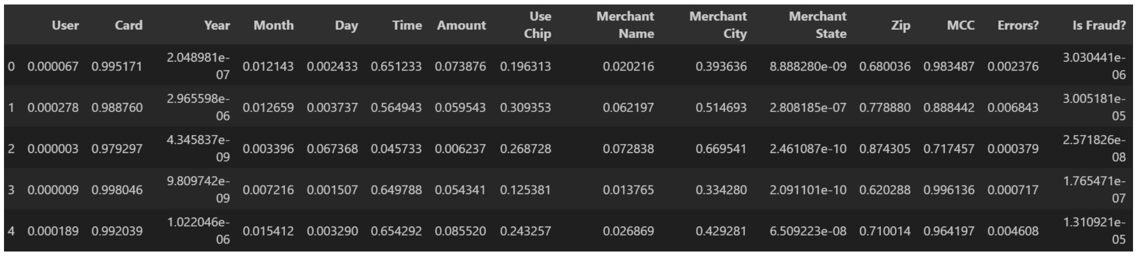
# 4 | Synthetic Data Generation using VAE Decoder

- Decoder model using the trained VAE model to generate synthetic data. It creates synthetic data points by sampling from the latent space and passing them through the decoder layers. The 'generate-synthetic-data' function allows adjusting the temperature parameter for controlling the diversity of the generated data. This approach enables the generation of new data points similar to the original dataset, useful for augmenting data or generating synthetic examples for testing models.

## 4.1 | Justification for Synthetic Data Generation

1. **Data Augmentation:** It allows for increasing the size of the dataset by generating new data points that follow the same distribution as the original data. This can be particularly useful when dealing with limited data availability.
2. **Diversity Exploration:** By adjusting the temperature parameter, the diversity of the synthetic data can be controlled. Higher temperatures result in more diverse data, enabling exploration of different scenarios and edge cases.
3. **Model Testing:**Synthetic data can be used for testing machine learning models, especially in cases where obtaining real-world data is challenging or costly. It helps in evaluating the model's generalization and robustness.
4. **Privacy Preservation:**Synthetic data generation can also be used for privacy-preserving data sharing. It allows sharing insights or models without exposing sensitive or personally identifiable information present in the original dataset.

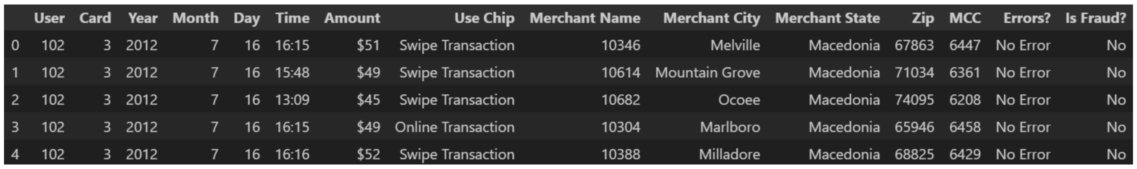
- Overall, leveraging VAE-based synthetic data generation enhances data-driven workflows by providing additional data points for training, testing, and exploration while maintaining the underlying data characteristics.



| | User | Card | Year | Month | Day | Time | Amount | Use Chip | Merchant Name | Merchant City | Merchant State | Zip | MCC | Errors? | Is Fraud? |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.000067 | 0.995171 | 2.048981e-07 | 0.012143 | 0.002433 | 0.651233 | 0.073876 | 0.196313 | 0.020216 | 0.393636 | 8.888280e-09 | 0.680036 | 0.983487 | 0.002376 | 3.030441e-06 |
| 1 | 0.000278 | 0.988760 | 2.965598e-06 | 0.012659 | 0.003737 | 0.564943 | 0.059543 | 0.309353 | 0.062197 | 0.514693 | 2.808185e-07 | 0.778880 | 0.888442 | 0.006843 | 3.005181e-05 |
| 2 | 0.000003 | 0.979297 | 4.345837e-09 | 0.003396 | 0.067368 | 0.045733 | 0.006237 | 0.268728 | 0.072838 | 0.669541 | 2.461087e-10 | 0.874305 | 0.717457 | 0.000379 | 2.571826e-08 |
| 3 | 0.000009 | 0.998046 | 9.809742e-09 | 0.007216 | 0.001507 | 0.649788 | 0.054341 | 0.125381 | 0.013765 | 0.334280 | 2.091101e-10 | 0.620288 | 0.996136 | 0.000717 | 1.765471e-07 |
| 4 | 0.000189 | 0.992039 | 1.022046e-06 | 0.015412 | 0.003290 | 0.654292 | 0.085520 | 0.243257 | 0.026869 | 0.429281 | 6.509223e-08 | 0.710014 | 0.964197 | 0.004608 | 1.310921e-05 |

**Figure 4.1:** Synthetic Data Generated before Denormalization

- **Denormalization of Data:** Process involves reverting the scaled or normalized data back to its original scale, which is essential for interpreting and using the data correctly. In this code snippet, the 'scaler.inverse-transform' function is applied to 'df-gen', which is assumed to be the original DataFrame before scaling. The resulting 'denormalizedDf' DataFrame contains the denormalized data.
- **Rounding Values in DataFrame:** Using 'applymap' along with a lambda function to round each value in the 'denormalizedDf' DataFrame. If the decimal part of a value is greater than or equal to 0.5, it is rounded up using 'np.ceil', otherwise it is rounded down using 'np.floor'. This operation ensures that the data is rounded to the nearest integer for each value in the DataFrame.
- **Decoding Categorical Data:** decodes categorical data columns using inverse transformations from previously defined encoders. It iterates through each column in the DataFrame 'data', checks if the column is encoded, and performs the inverse transformation to retrieve the original labels. The resulting DataFrame 'denormalizedDf-Labeled' contains the decoded categorical columns along with the unchanged numerical columns.



| | User | Card | Year | Month | Day | Time | Amount | Use Chip | Merchant Name | Merchant City | Merchant State | Zip | MCC | Errors? | Is Fraud? |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 102 | 3 | 2012 | 7 | 16 | 16:15 | $51 | Swipe Transaction | 10346 | Melville | Macedonia | 67863 | 6447 | No Error | No |
| 1 | 102 | 3 | 2012 | 7 | 16 | 15:48 | $49 | Swipe Transaction | 10614 | Mountain Grove | Macedonia | 71034 | 6361 | No Error | No |
| 2 | 102 | 3 | 2012 | 7 | 16 | 13:09 | $45 | Swipe Transaction | 10682 | Ocoee | Macedonia | 74095 | 6208 | No Error | No |
| 3 | 102 | 3 | 2012 | 7 | 16 | 16:15 | $49 | Online Transaction | 10304 | Marlboro | Macedonia | 65946 | 6458 | No Error | No |
| 4 | 102 | 3 | 2012 | 7 | 16 | 16:16 | $52 | Swipe Transaction | 10388 | Milladore | Macedonia | 68825 | 6429 | No Error | No |

**Figure 4.2:** Final Synthetic Data Generated

## 4.2 | KL Divergence Analysis

- a function 'kl-divergence' to calculate the Kullback-Leibler (KL) divergence between two probability distributions. The 'calculate-kl-divergence' function computes the KL divergence for each numerical column in two DataFrames 'df1' and 'df2', assuming they are preprocessed and scaled. The resulting KL divergences are then visualized using a bar chart, showcasing the divergence between the distributions of the two datasets across different columns.

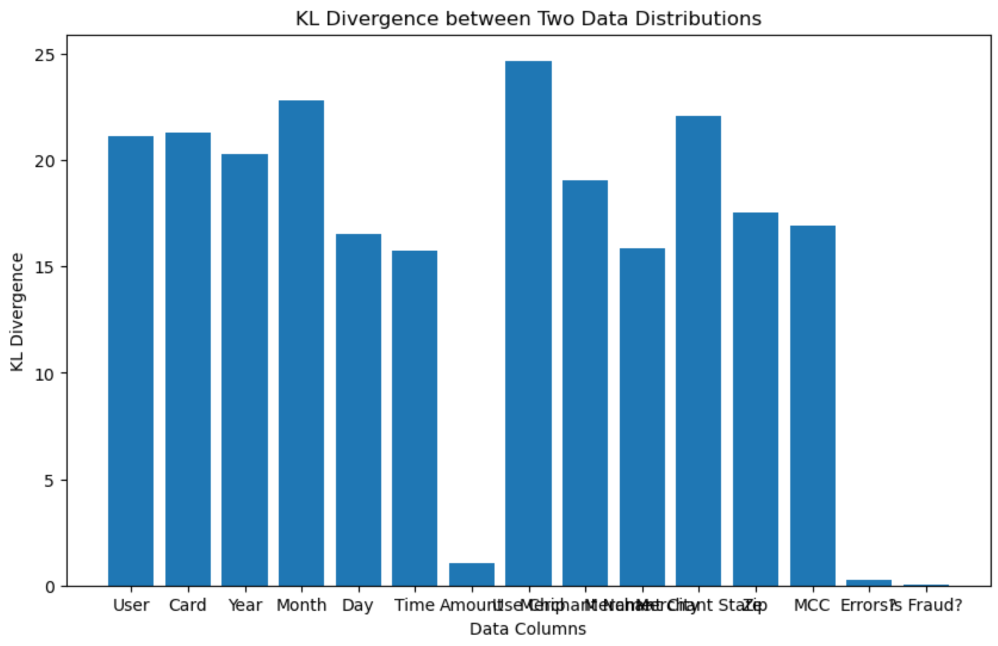


**Figure 4.3:** KL Divergence between Two Data Distributions

## 4.3 Evaluation

1. **Coarse grained:** The metric help gauge the level of similarity and uniqueness between the real and synthetic datasets, providing insights into the quality and diversity of the synthetic data generation process.

   - It uses two foundations for evaluation:

     [a] **Direct Copy Percentage:** It calculates what percentage of the combined (real and synthetic) data is a direct copy of the real data.

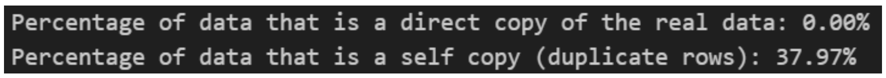
     $$\text{Direct Copy Percentage} = \frac{\text{Total Data Size}}{\text{Number of Copies}} \times 100$$

     Here, ”copies” refer to rows in the combined data that are not present in either the real or synthetic datasets, indicating direct replication.

     [b] **Self-Copy Percentage:** It measures the percentage of duplicate rows (self-copies) within the synthetic data. This metric reveals how much of the synthetic data consists of repeated patterns or copies of existing rows within itself.

     $$\text{Self-Copy Percentage} = \frac{\text{Number of Duplicate Rows in Synthetic Data}}{\text{Total Synthetic Data Size}} \times 100$$

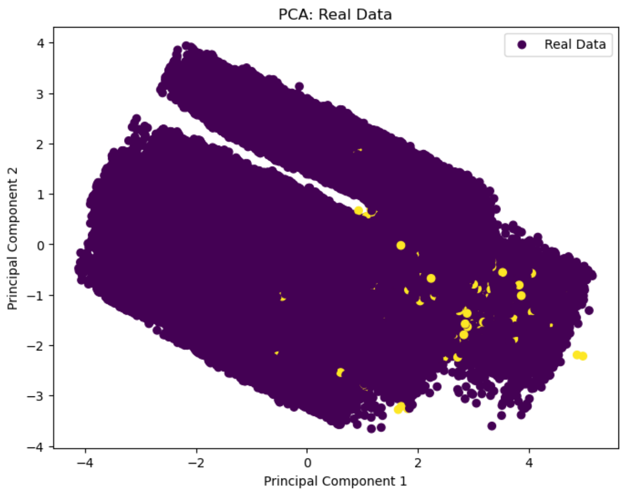
   - Output of Coarse grained evaluation:



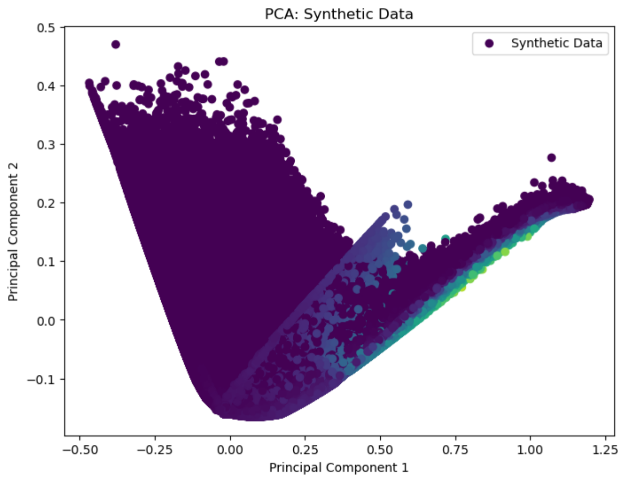
```
Percentage of data that is a direct copy of the real data: 0.00%
Percentage of data that is a self copy (duplicate rows): 37.97%
```

**Figure 4.4:** Coarse grained evaluation

2. **Medium grained:** This evaluation metric compares the distributions of real and synthetic data. This evaluation provides insights into how well the synthetic data replicates the distribution patterns of the real data across various features. By visualizing the distributions side by side, it helps assess the overall similarity or dissimilarity between the two datasets in terms of their statistical properties. This comparison can be crucial for understanding whether the synthetic data generation process accurately captures the underlying characteristics of the real data distribution.



**(a)** PCA Real Data



**(b)** PCA Synthetic Data

**Figure 4.5:** Medium Grained Evaluation

3. **Fine grained:** This evaluation metric involves comparing the joined distributions of real and synthetic data. This metric goes beyond comparing individual feature distributions and focuses on the overall joint distribution of multiple features. By analyzing how well the joint distributions align between real and synthetic data, this evaluation provides a deeper understanding of the data generation process's fidelity. It helps identify any discrepancies or patterns that may not be apparent when examining individual features separately, thus offering a more comprehensive assessment of the synthetic data's quality and resemblance to real-world data.

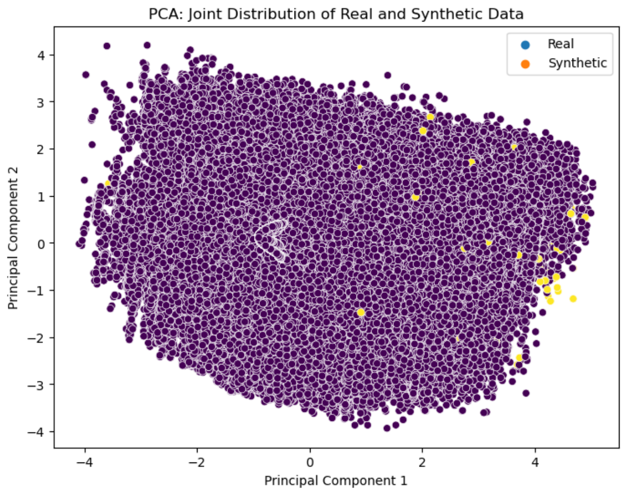


**Figure 4.6:** PCA: Joint Distribution of Real and Synthetic Data