**Problem Statement:**

Generate dummy data using any library and generate dummy data using any generative model train two different models and compare their accuracies  
and compare accuracy of model on data generated by library and data generated by generative model.

**Formulate Problem:**

The problem is to generate a dummy dataset for disease diagnosis prediction using a GAN model and using numpy library.The dataset consist of 1000 rows and 10 columns.

**Generative Adversial network:**

This report presents a dataset that has been scaled and transformed for use in a Generative Adversarial Network (GAN). The GAN model consists of a **Generator** and a **Discriminator**, both implemented using PyTorch. The goal is to generate synthetic data resembling the original dataset.

**2. Data Preprocessing**

* The dataset is first normalized using **MinMaxScaler**, which scales all values between **0 and 1** to ensure consistent input for the neural network.
* The scaled data is converted into a **PyTorch tensor (torch.tensor)** to be compatible with deep learning models.

**3. Model Architecture**

**Generator Model:**

* The Generator takes random noise (noise\_dim) as input.
* It passes through a **fully connected neural network** with one hidden layer of **16 neurons** and ReLU activation.
* The output layer applies a **Sigmoid activation** to ensure values remain between **0 and 1**, matching the scaled data distribution.

**Discriminator Model:**

* The Discriminator takes real or fake data as input (input\_dim).
* It consists of a **fully connected neural network** with **16 neurons** in the hidden layer and ReLU activation.
* The output layer uses a **Sigmoid activation** to predict whether the input data is real or fake.

**Model Setup**

* **Noise Dimension (noise\_dim)**: Set to **10**, representing the random input for the Generator.
* **Input Dimension (input\_dim)**: Matches the number of features in the scaled dataset.
* **Batch Size (batch\_size)**: **32** samples per training step.
* **Number of Epochs (num\_epochs)**: **5000** iterations to ensure sufficient learning.
* **Learning Rate (lr)**: **0.001**, optimized using the **Adam** optimizer.

**3. Training Process**

Training is performed using a **mini-batch approach** with the following key steps:

1. **Data Loading**
   * The scaled dataset is converted into **PyTorch tensors** and loaded using DataLoader for efficient batch processing.
2. **Discriminator Training**
   * The Discriminator is trained on both **real** and **fake** data.
   * **Real Data**: Extracted from the dataset and assigned label **1 (real)**.
   * **Fake Data**: Generated by the Generator from random noise and assigned label **0 (fake)**.

**Loss Calculation**: Binary Cross-Entropy Loss (BCEWithLogitsLoss) is used to measure how well the Discriminator differentiates between real and fake data.

* + **Optimization**: Gradients are computed, and the Discriminator weights are updated using **Adam optimizer**.

1. **Generator Training**
   * A new batch of **fake data** is generated using random noise.
   * The Discriminator evaluates this fake data, and the **Generator is rewarded when the Discriminator classifies fake data as real**.
   * **Loss Calculation**: The Generator’s goal is to minimize the Discriminator’s ability to distinguish real from fake data, optimizing against real\_labels.
   * **Optimization**: Gradients are computed, and the Generator weights are updated.
2. **Progress Monitoring**
   * Training loss values for the Discriminator (D Loss) and Generator (G Loss) are printed every **500 epochs** to track performance.

**4. Expected Outcome**

* Initially, the **Discriminator performs well** at distinguishing real from fake data.
* Over time, as the Generator improves, the fake data becomes more realistic, making it difficult for the Discriminator to differentiate.
* At equilibrium, both the Discriminator and Generator should reach a **stable state** where fake data closely resembles real data.

**Visualization**:

* + A **Kernel Density Estimation (KDE) plot** is created to compare the distribution of **Age** in the real vs fake data. This visualization helps to observe how closely the fake data matches the real data in terms of feature distribution.

**3. Model Training and Evaluation:**

* **Train-Test Split** :Both the real and fake datasets are split into training and testing sets (80% training, 20% testing) for both features (X) and labels (y).
* **Logistic Regression**:
  + Logistic Regression models are trained on both the real and fake datasets and evaluated based on accuracy. The accuracy measures how well the model can predict the **Diagnosis Label**.
* **Random Forest Classifier**:
  + Random Forest models (with 100 estimators) are also trained on both datasets and evaluated using accuracy scores. This provides a robust comparison of classifier performance.

**4. Results**

* **Accuracy Scores**:  
  The results show the accuracy of both classifiers on the real and fake datasets:

**Logistic Regression:**

* + **Accuracy on Real Data**: 0.5050
  + **Accuracy on Fake Data**: 0.8450

**Random Forest:**

* + **Accuracy on Real Data (Random Forest)**:0.4400
  + **Accuracy on Fake Data (Random Forest)**: 0.9250

These accuracy scores highlight how well each model performs on both the real and synthetic data, with higher values indicating better model generalization.

* **Visualization of Model Performance**:  
  A bar plot compares the **accuracy** of both **Logistic Regression** and **Random Forest** models across the real and fake datasets. The models are grouped by real or fake data, and color coding is used for easy comparison:
  + **Blue**: Logistic Regression on real data
  + **Red**: Logistic Regression on fake data
  + **Green**: Random Forest on real data
  + **Orange**: Random Forest on fake data

This comparison allows for an immediate visual assessment of how the models behave with both types of data.