a. import numpy as np

import pandas as pd

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import confusion\_matrix, classification\_report

import matplotlib.pyplot as plt

from tensorflow.keras import layers, models

b. dataset = pd.read\_csv("creditcard.csv")

c. dataset.size

d. scaler = StandardScaler()

X = scaler.fit\_transform(dataset.drop("Class", axis=1))

y = dataset["Class"]

e. # Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

f. # Build and train the Autoencoder model

input\_dim = X\_train.shape[1]

g. # Encoder

encoder = models.Sequential([

layers.Input(shape=(input\_dim,)),

layers.Dense(32, activation='relu'),

layers.Dense(16, activation='relu')

])

h. # Decoder

decoder = models.Sequential([

layers.Input(shape=(16,)),

layers.Dense(32, activation='relu'),

layers.Dense(input\_dim, activation='linear') # Using 'linear' activation for reconstruction

])

# Autoencoder

autoencoder = models.Sequential([

encoder,

decoder

])

1. autoencoder.compile(optimizer='adam', loss='mean\_squared\_error')

autoencoder.fit(X\_train, X\_train, epochs=10, batch\_size=32, shuffle=True, validation\_data=(X\_test, X\_test))

j. # Detect anomalies and tune the threshold

y\_pred = autoencoder.predict(X\_test)

mse = np.mean(np.power(X\_test - y\_pred, 2), axis=1)

k. # Visualize the reconstruction error distribution

plt.figure(figsize=(10, 6))

plt.hist(mse, bins=50, alpha=0.5, color='b', label='Reconstruction Error')

plt.xlabel("Reconstruction Error")

plt.ylabel("Frequency")

plt.legend()

plt.title("Reconstruction Error Distribution")

plt.show()

l. # Threshold tuning (iterate and adjust as needed)

thresholds = np.arange(0.1, 1.0, 0.1) # Adjust the step size as needed

for threshold in thresholds:

anomalies = mse > threshold

plt.show()

m. # Count the number of anomalies

num\_anomalies = np.sum(anomalies)

print(f"Threshold: {threshold:.1f}, Number of anomalies: {num\_anomalies}")

n. # Evaluate the model

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, anomalies))

print("\nClassification Report:")

print(classification\_report(y\_test, anomalies))

o. import seaborn as sns

p. plt.figure(figsize = (6, 4.75))

sns.heatmap(confusion\_matrix(y\_test, anomalies), annot = True, annot\_kws = {"size": 16}, fmt = 'd')

plt.xticks([0.5, 1.5], rotation = 'horizontal')

plt.yticks([0.5, 1.5], rotation = 'horizontal')

plt.xlabel("Predicted label", fontsize = 14)

plt.ylabel("True label", fontsize = 14)

plt.title("Confusion Matrix", fontsize = 14)

plt.grid(False)

plt.show()