**Anomaly Detection System: Program Description and Testing**

This document explains two Python programs: anomaly\_ui.py (the main program) and anomalytestscript.py (the testing script). These programs work together to simulate a bank transaction system that uses artificial intelligence (AI) to spot unusual activities—like someone spending $10,000 when they usually spend $500. The main program includes a webpage dashboard, while the testing script ensures everything works correctly.

**Part 1: Main Program (anomaly\_ui.py)**

This program creates a system to detect unusual bank transactions using AI. Think of it as a smart bank clerk who flags suspicious activities and shows you the results on a webpage.

**What the Program Does**

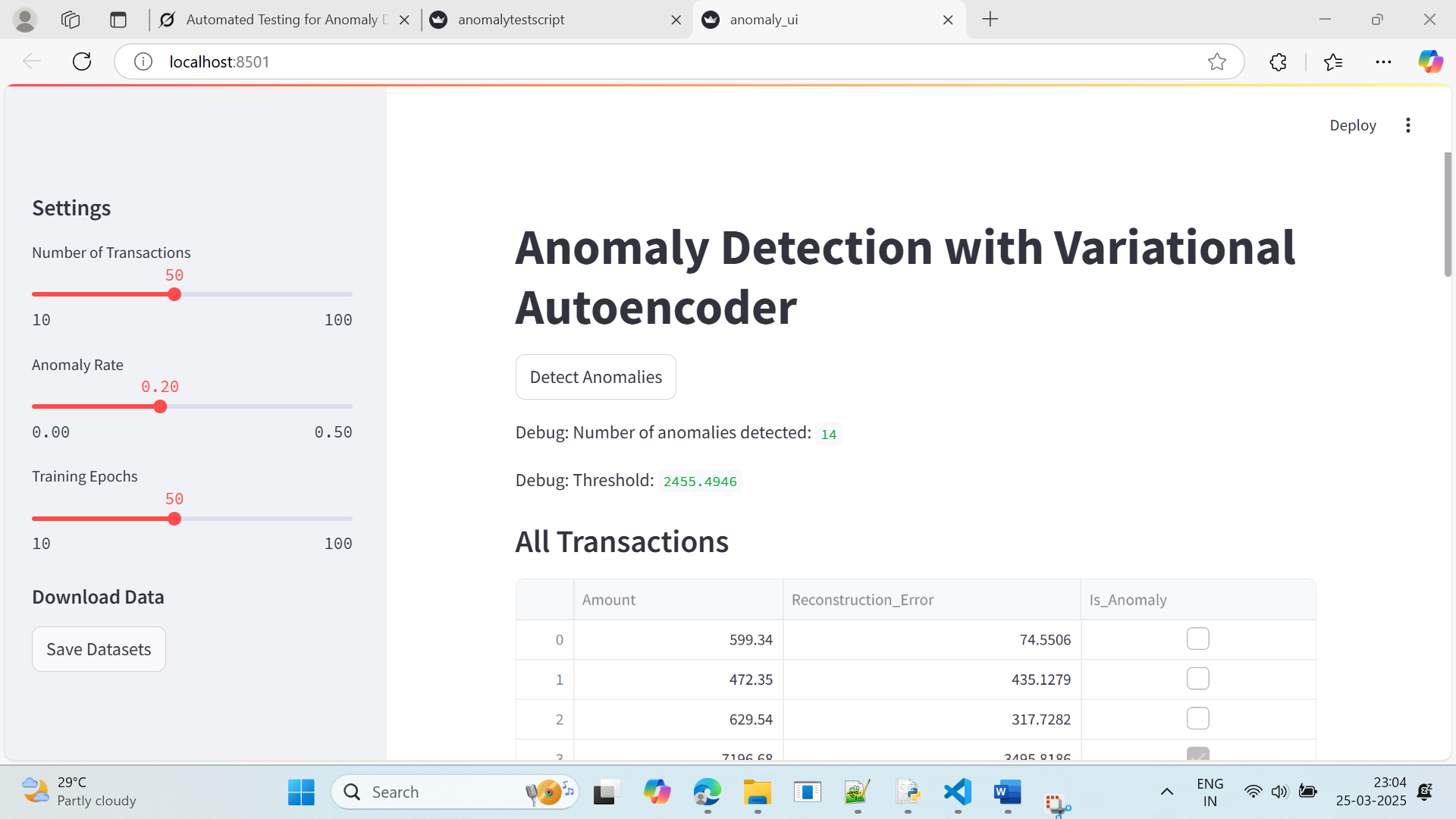
* **Creates Fake Bank Records**: Generates pretend transactions to test the system.
* **Uses AI to Find Odd Transactions**: Trains an AI to spot unusual patterns.
* **Shows Results on a Webpage**: Displays tables and graphs in a browser.
* **Lets You Save Data**: Allows downloading the fake records as files.

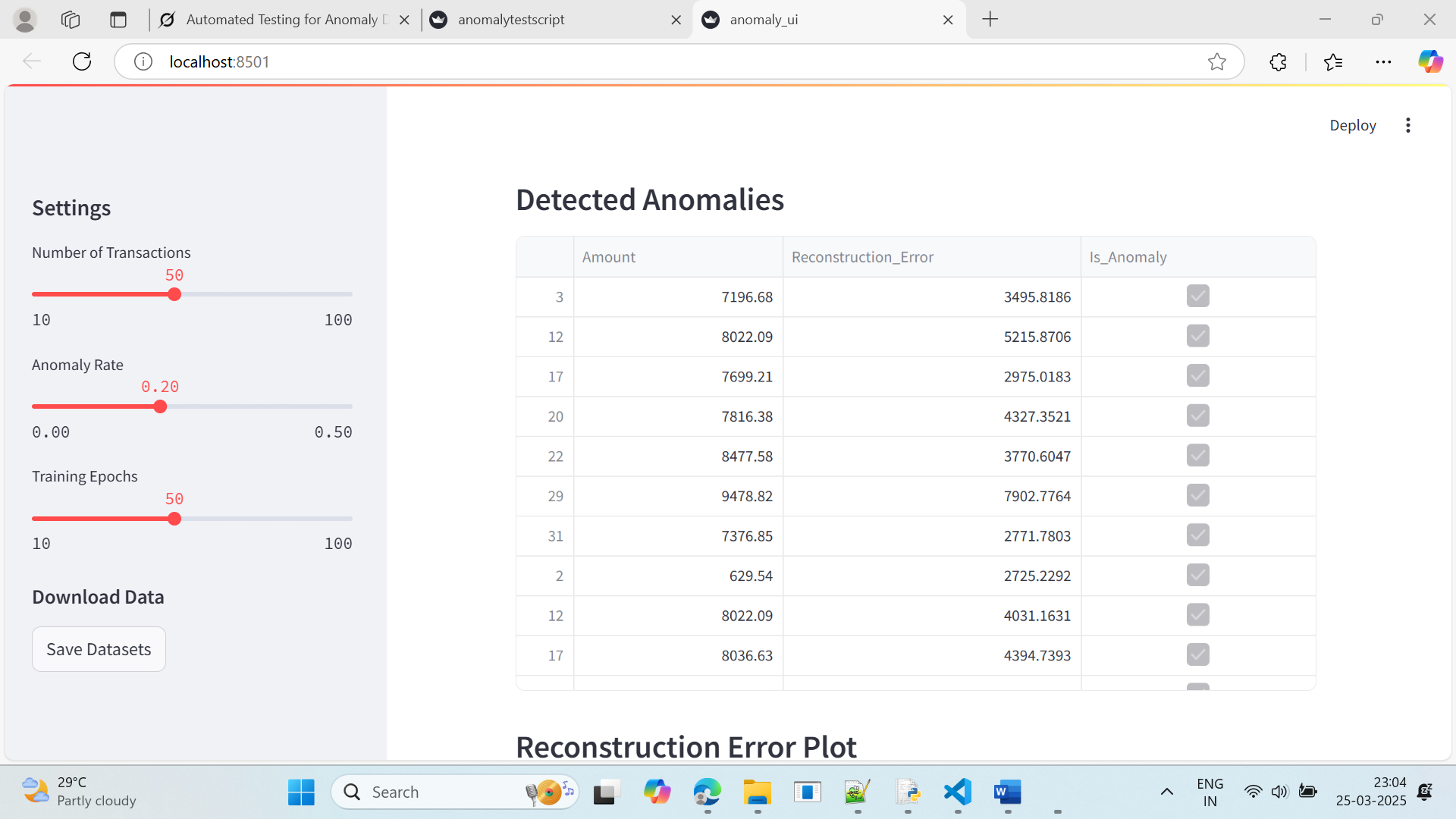
It’s like a toy bank simulator with a built-in detective!

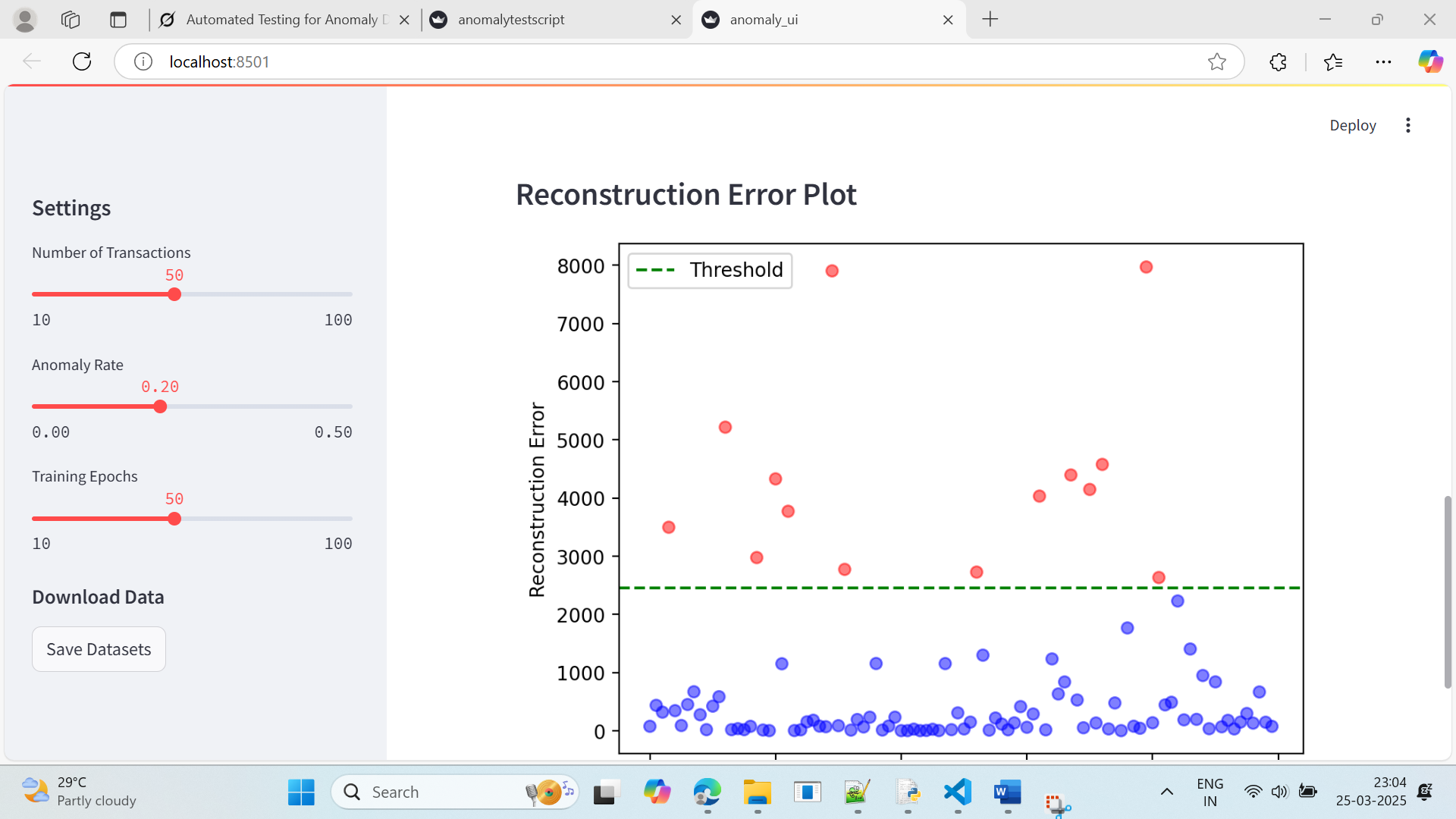
anomaly\_ui.py

Used Grok AI tool to create this tool :

**Output of the Program :**







Let’s break down this Python program in simple terms, as if I’m explaining it to someone who’s never coded before. Imagine this program as a smart bank clerk who uses artificial intelligence (AI) to spot unusual transactions—like someone suddenly spending $10,000 when they usually spend $500. The program also has a little dashboard you can play with in your web browser. Here’s how it works, step by step.

**How It Works: Step by Step**

**1. Tools We Need (Imports)**

import pandas as pd           # Organizes data like a spreadsheet

import numpy as np           # Handles random numbers

from datetime import datetime, timedelta  # Manages dates

import torch                 # Powers the AI

import torch.nn as nn        # Builds the AI’s brain

import torch.optim as optim  # Helps the AI learn

import streamlit as st       # Creates a webpage

import matplotlib.pyplot as plt  # Draws graphs

**2. Making Fake Bank Records**

**The generate\_synthetic\_data function creates pretend transactions:**

np.random.seed(42)  # Ensures consistent random numbers

def generate\_synthetic\_data(num\_records, base\_date, anomaly\_rate=0.2):

    transaction\_ids = range(1, num\_records + 1)  # IDs: 1, 2, 3, ...

    dates = [base\_date + timedelta(days=i) for i in range(num\_records)]  # Dates: March 1, March 2, ...

    amounts = np.random.normal(loc=500, scale=200, size=num\_records).clip(min=50, max=1000)  # Amounts around $500

    num\_anomalies = int(num\_records \* anomaly\_rate)  # 20% are odd

    anomaly\_indices = np.random.choice(num\_records, num\_anomalies, replace=False)

    amounts[anomaly\_indices] = np.random.uniform(5000, 10000, num\_anomalies)  # Odd amounts: $5000-$10,000

    data = pd.DataFrame({

        'Transaction\_ID': transaction\_ids,

        'Date': dates,

        'Amount': amounts

    })

    return data

* **What Happens**:
  + Creates a set number of transactions (e.g., 50).
  + Assigns IDs and dates starting from a base date (e.g., March 1, 2025).
  + Sets most amounts around $500, but ensures they’re between $50 and $1000.
  + Makes 20% of them “weird” by setting amounts between $5000 and $10,000.
  + Returns the data as a table.

**3. Building the AI Detective (VAE)**

The VAE class is the AI brain, called a Variational Autoencoder:

class VAE(nn.Module):

    def \_\_init\_\_(self, input\_dim, hidden\_dim, latent\_dim):

        super(VAE, self).\_\_init\_\_()

        self.encoder = nn.Sequential(nn.Linear(input\_dim, hidden\_dim), nn.ReLU(), nn.Linear(hidden\_dim, latent\_dim \* 2))

        self.decoder = nn.Sequential(nn.Linear(latent\_dim, hidden\_dim), nn.ReLU(), nn.Linear(hidden\_dim, input\_dim))

    def reparameterize(self, mu, logvar):

        std = torch.exp(0.5 \* logvar)

        eps = torch.randn\_like(std)

        return mu + eps \* std

    def forward(self, x):

        h = self.encoder(x)

        mu, logvar = h.chunk(2, dim=1)

        z = self.reparameterize(mu, logvar)

        recon\_x = self.decoder(z)

        return recon\_x, mu, logvar

* **What It Does**:
  + Takes transaction amounts and summarizes them (encoder).
  + Rebuilds the amounts from the summary (decoder).
  + Adds randomness (reparameterize) to help the AI learn patterns.
  + The forward function runs this process.

**4. Scoring the AI’s Work**

The vae\_loss function grades how well the AI rebuilds the data:

def vae\_loss(recon\_x, x, mu, logvar):

    recon\_loss = nn.functional.mse\_loss(recon\_x, x, reduction='sum')  # How close is the rebuild?

    kl\_div = -0.5 \* torch.sum(1 + logvar - mu.pow(2) - logvar.exp())  # Prevents cheating

    return recon\_loss + kl\_div

* **What Happens**:
  + Compares the rebuilt amounts to the originals (recon\_loss).
  + Adds a penalty (kl\_div) to ensure the AI learns general patterns, not just memorizes.

**5. Training the AI**

The train\_vae function teaches the AI to spot odd transactions:

def train\_vae(data, epochs=50):

    min\_val = float(data['Amount'].min())

    max\_val = float(data['Amount'].max())

    normalized\_data = (data - min\_val) / (max\_val - min\_val)  # Scale to 0-1

    input\_dim = normalized\_data.shape[1]

    vae = VAE(input\_dim=input\_dim, hidden\_dim=16, latent\_dim=8)

    optimizer = optim.Adam(vae.parameters(), lr=1e-3)

    data\_tensor = torch.FloatTensor(normalized\_data.values)

    for epoch in range(epochs):

        vae.zero\_grad()

        recon\_data, mu, logvar = vae(data\_tensor)

        loss = vae\_loss(recon\_data, data\_tensor, mu, logvar)

        loss.backward()

        optimizer.step()

    with torch.no\_grad():

        recon\_data, \_, \_ = vae(data\_tensor)

        recon\_error = torch.mean((recon\_data - data\_tensor) \*\* 2, dim=1)

        threshold = recon\_error.mean() + 1 \* recon\_error.std()

        anomalies = recon\_error > threshold

    recon\_error = recon\_error \* (max\_val - min\_val)

    return anomalies.numpy(), recon\_error.numpy(), min\_val, max\_val

* **What Happens**:
  + Scales amounts to a 0-1 range for easier learning.
  + Creates the VAE with specific sizes (input\_dim, hidden\_dim=16, latent\_dim=8).
  + Trains it for a set number of rounds (epochs, e.g., 50).
  + Measures how far off the rebuilt amounts are (reconstruction error).
  + Sets a threshold for “weirdness” (average error + 1 standard deviation).
  + Flags transactions above the threshold as anomalies.
  + Returns the anomaly flags, errors, and scaling values.

**6. Creating the Web Dashboard**

The main function sets up an interactive webpage using Streamlit:

def main():

    st.title("Anomaly Detection with Variational Autoencoder")

    st.sidebar.header("Settings")

    num\_records = st.sidebar.slider("Number of Transactions", 10, 100, 50)

    anomaly\_rate = st.sidebar.slider("Anomaly Rate", 0.0, 0.5, 0.2, step=0.05)

    epochs = st.sidebar.slider("Training Epochs", 10, 100, 50, step=10)

    base\_date = datetime(2025, 3, 1)

    data1 = generate\_synthetic\_data(num\_records, base\_date, anomaly\_rate)

    data2 = data1.copy()

    discrepancy\_indices = np.random.choice(num\_records, int(num\_records \* 0.2), replace=False)

    data2.loc[discrepancy\_indices, 'Amount'] = data2.loc[discrepancy\_indices, 'Amount'] \* np.random.uniform(0.8, 1.2, len(discrepancy\_indices))

    data1['Amount'] = data1['Amount'].round(2)

    data2['Amount'] = data2['Amount'].round(2)

    combined\_data = pd.concat([data1[['Amount']], data2[['Amount']]], axis=0)

    if st.button("Detect Anomalies"):

        with st.spinner("Training VAE and detecting anomalies..."):

            anomalies, recon\_errors, min\_val, max\_val = train\_vae(combined\_data, epochs=epochs)

            combined\_data['Reconstruction\_Error'] = recon\_errors

            combined\_data['Is\_Anomaly'] = anomalies

            st.write("Debug: Number of anomalies detected:", anomalies.sum())

            st.write("Debug: Threshold:", (recon\_errors.mean() + 1 \* recon\_errors.std()))

            st.subheader("All Transactions")

            st.write(combined\_data)

            st.subheader("Detected Anomalies")

            anomaly\_data = combined\_data[combined\_data['Is\_Anomaly']]

            if not anomaly\_data.empty:

                st.write(anomaly\_data)

            else:

                st.write("No anomalies detected. Try increasing anomaly rate or lowering threshold.")

            st.subheader("Reconstruction Error Plot")

            fig, ax = plt.subplots()

            ax.scatter(range(len(recon\_errors)), recon\_errors, c=['red' if a else 'blue' for a in anomalies], alpha=0.5)

            ax.axhline(y=recon\_errors.mean() + 1 \* recon\_errors.std(), color='green', linestyle='--', label='Threshold')

            ax.set\_xlabel("Transaction Index")

            ax.set\_ylabel("Reconstruction Error")

            ax.legend()

            st.pyplot(fig)

    st.sidebar.subheader("Download Data")

    if st.sidebar.button("Save Datasets"):

        data1.to\_csv('synthetic\_bank\_statement1.csv', index=False)

        data2.to\_csv('synthetic\_bank\_statement2.csv', index=False)

        st.sidebar.success("Datasets saved as 'synthetic\_bank\_statement1.csv' and 'synthetic\_bank\_statement2.csv'")

* **What Happens**:
  + **Title and Settings**: Displays a title and sidebar sliders to set:
    - Number of transactions (10 to 100, default 50).
    - Anomaly rate (0% to 50%, default 20%).
    - Training rounds (10 to 100, default 50).
  + **Data Creation**: Generates two sets of fake data:
    - data1: Original transactions.
    - data2: Copies data1 but tweaks 20% of amounts slightly (80%-120% of original).
    - Combines them for the AI to analyze.
  + **Detect Anomalies Button**:
    - Trains the AI and flags anomalies.
    - Shows debug info (number of anomalies, threshold).
    - Displays tables for all transactions and just the anomalies.
    - Draws a graph (blue dots for normal, red for anomalies, green line for threshold).
  + **Download Option**: Saves the datasets as CSV files when the “Save Datasets” button is clicked.

**7. Starting the Program**

if \_\_name\_\_ == "\_\_main\_\_":

    main()

* **Runs the program and opens the webpage.**

**Running the Program**

**Run this command in your terminal:**

& C:/Users/mahes/AppData/Local/Microsoft/WindowsApps/python3.13.exe -m streamlit run C:\Users\mahes\OneDrive\Desktop\anomalytestscript.py

* **It opens a webpage (e.g.,** [**http://localhost:8501**](http://localhost:8501)**) where you can interact with the system.**

**Why It’s Useful**

* **Practice Tool: It’s a safe way to test an AI for spotting weird bank transactions without using real money.**
* **Easy to Use: The webpage makes it fun and simple to explore the results.**
* **Learning Opportunity: Great for understanding how AI can detect anomalies in data.**

**---------------------------PART 2 Testing ------------------------------------**

**Part 2: Testing Script (anomalytestscript.py)**

**This script tests the main program to ensure it works correctly. Think of it as a teacher checking a student’s homework to confirm all answers are right.**

**What the Testing Script Does**

* **Tests Four Key Parts:** 
  + **Does the fake data generator work?**
  + **Is the AI brain built correctly?**
  + **Can the AI find odd transactions?**
  + **Can the AI say “nothing weird” when there are no anomalies?**
* **Custom Report: Shows “4 passed” if all tests succeed, instead of just “OK.”**

**How It Works: Step by Step**

**1. Tools We Need (Imports)**

import unittest

import pandas as pd

import numpy as np

from datetime import datetime, timedelta

import torch

import torch.nn as nn

from unittest.mock import patch

import io

import sys

* **Tools Explained:** 
  + **unittest: Creates a checklist for testing.**
  + **pandas, numpy, datetime, timedelta, torch, nn: Same as the main program (for data and AI).**
  + **patch, io, sys: Help capture and test the program’s messages.**

**2. Making Fake Bank Data**

**This is the same generate\_synthetic\_data function as in the main program:**

* **Creates fake transactions with IDs, dates, and amounts.**
* **Adds some big, odd amounts (e.g., $5000-$10,000) for testing.**

**3. Building the AI Brain (VAE)**

**The same VAE class as in the main program:**

* **Summarizes and rebuilds data to learn patterns.**
* **Used to test if the AI structure is correct.**

**4. Scoring the AI’s Work**

**The same vae\_loss function:**

* **Grades the AI’s rebuilding accuracy.**
* **Used during training to test the AI’s learning.**

**5. Training the AI**

**The train\_vae function, slightly modified for testing:**

def train\_vae(data, epochs=50, threshold\_multiplier=1.0):

    min\_val = float(data['Amount'].min())

    max\_val = float(data['Amount'].max())

    normalized\_data = (data - min\_val) / (max\_val - min\_val)

    input\_dim = normalized\_data.shape[1]

    vae = VAE(input\_dim=input\_dim, hidden\_dim=16, latent\_dim=8)

    optimizer = torch.optim.Adam(vae.parameters(), lr=1e-3)

    data\_tensor = torch.FloatTensor(normalized\_data.values)

    for epoch in range(epochs):

        vae.zero\_grad()

        recon\_data, mu, logvar = vae(data\_tensor)

        loss = vae\_loss(recon\_data, data\_tensor, mu, logvar)

        loss.backward()

        optimizer.step()

    with torch.no\_grad():

        recon\_data, \_, \_ = vae(data\_tensor)

        recon\_error = torch.mean((recon\_data - data\_tensor) \*\* 2, dim=1)

        threshold = recon\_error.mean() + threshold\_multiplier \* recon\_error.std()

        anomalies = recon\_error > threshold

    recon\_error = recon\_error \* (max\_val - min\_val)

    return anomalies.numpy(), recon\_error.numpy(), min\_val, max\_val

* **What’s New:** 
  + **Added threshold\_multiplier to adjust the “weirdness” threshold during tests.**

**6. Custom Test Report**

**The testing script customizes how results are shown:**

class CustomTestResult(unittest.TextTestResult):

    def printErrors(self):

        if self.errors or self.failures:

            super().printErrors()

    def addSuccess(self, test):

        super().addSuccess(test)

    def wasSuccessful(self):

        return len(self.failures) == 0 and len(self.errors) == 0

    def print\_result(self):

        if self.wasSuccessful():

            print(f"{self.testsRun} passed")

        else:

            print(f"FAILED (failures={len(self.failures)}, errors={len(self.errors)})")

class CustomTestRunner(unittest.TextTestRunner):

    def \_\_init\_\_(self, \*args, \*\*kwargs):

        super().\_\_init\_\_(resultclass=CustomTestResult, \*args, \*\*kwargs)

    def run(self, test):

        result = super().run(test)

        result.print\_result()

        return result

* **What Happens:** 
  + **If all tests pass, it prints “4 passed” (or however many tests there are).**
  + **If any fail, it shows the number of failures and errors.**

**7. The Tests**

**The TestAnomalyDetection class runs four tests:**

* **Test 1: test\_generate\_synthetic\_data:** 
  + **Creates fake data and checks:** 
    - **Correct number of rows and columns.**
    - **IDs and dates are correct.**
    - **Some amounts are big (anomalies).**
* **Test 2: test\_vae\_model:** 
  + **Builds the VAE and tests if it processes data correctly (sizes match).**
* **Test 3: test\_train\_vae:** 
  + **Trains the VAE with fake data and checks if it finds anomalies.**
* **Test 4: test\_no\_anomalies\_output:** 
  + **Uses normal data (no big amounts) and checks if the AI says “no anomalies.”**

**8. Running the Tests**

if \_\_name\_\_ == '\_\_main\_\_':

    suite = unittest.TestLoader().loadTestsFromTestCase(TestAnomalyDetection)

    CustomTestRunner(verbosity=2).run(suite)

* **Loads and runs all four tests, showing detailed results.**

**Running the Tests**

**Run this command:**

**& C:/Users/mahes/AppData/Local/Microsoft/WindowsApps/python3.13.exe C:\Users\mahes\OneDrive\Desktop\anomalytestscript.py**

* **Output: If all tests pass, you’ll see “4 passed” in the terminal. (Note: The document mentions a screenshot of the output, but it’s not included here.)**

**Why It’s Useful**

* **Safety Net: Ensures the main program works correctly.**
* **Confidence: If someone changes the code later, these tests confirm it still functions.**
* **For Beginners: Think of it as a teacher grading homework—making sure every step is right before saying “good job!”**

