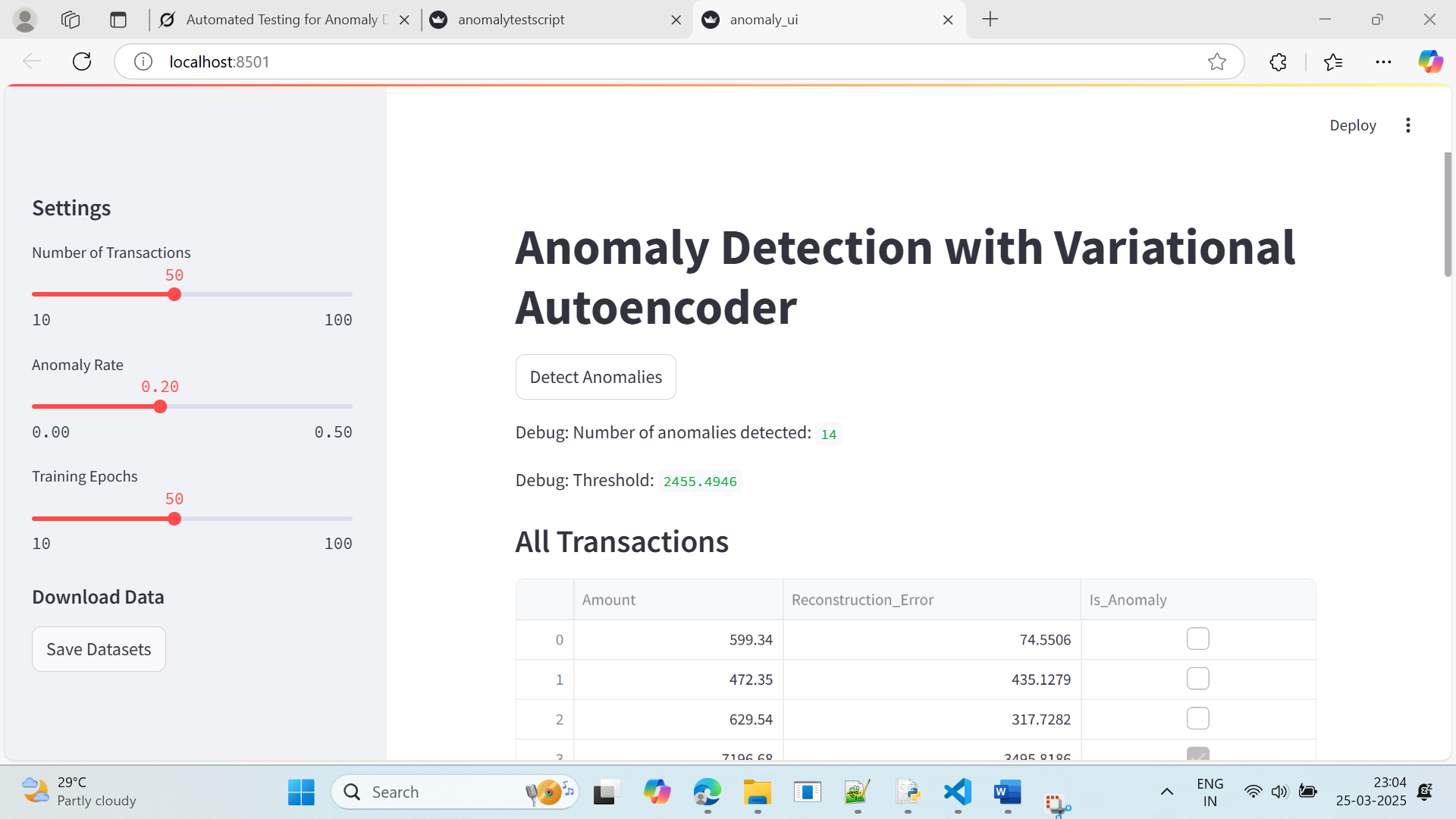
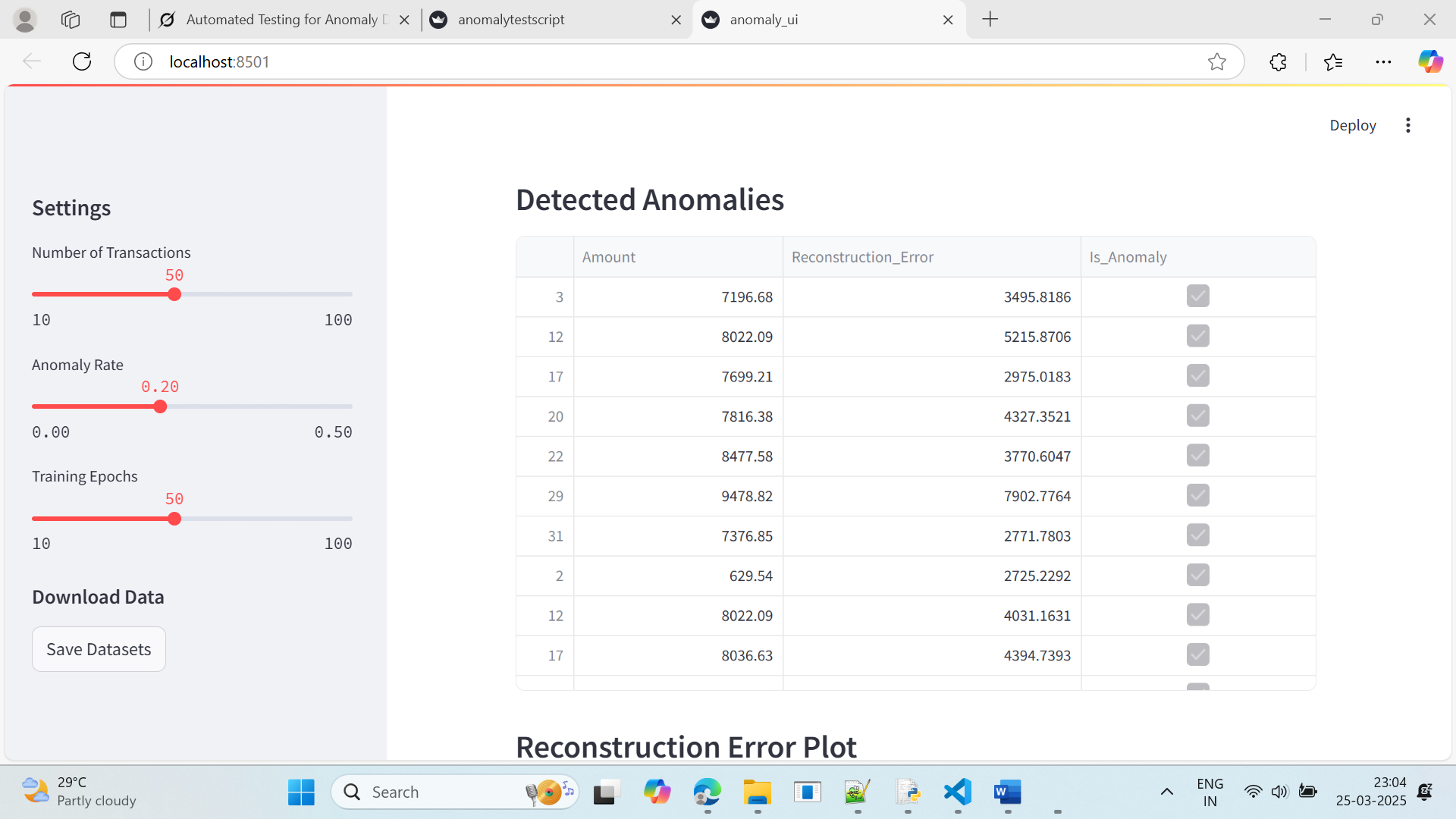
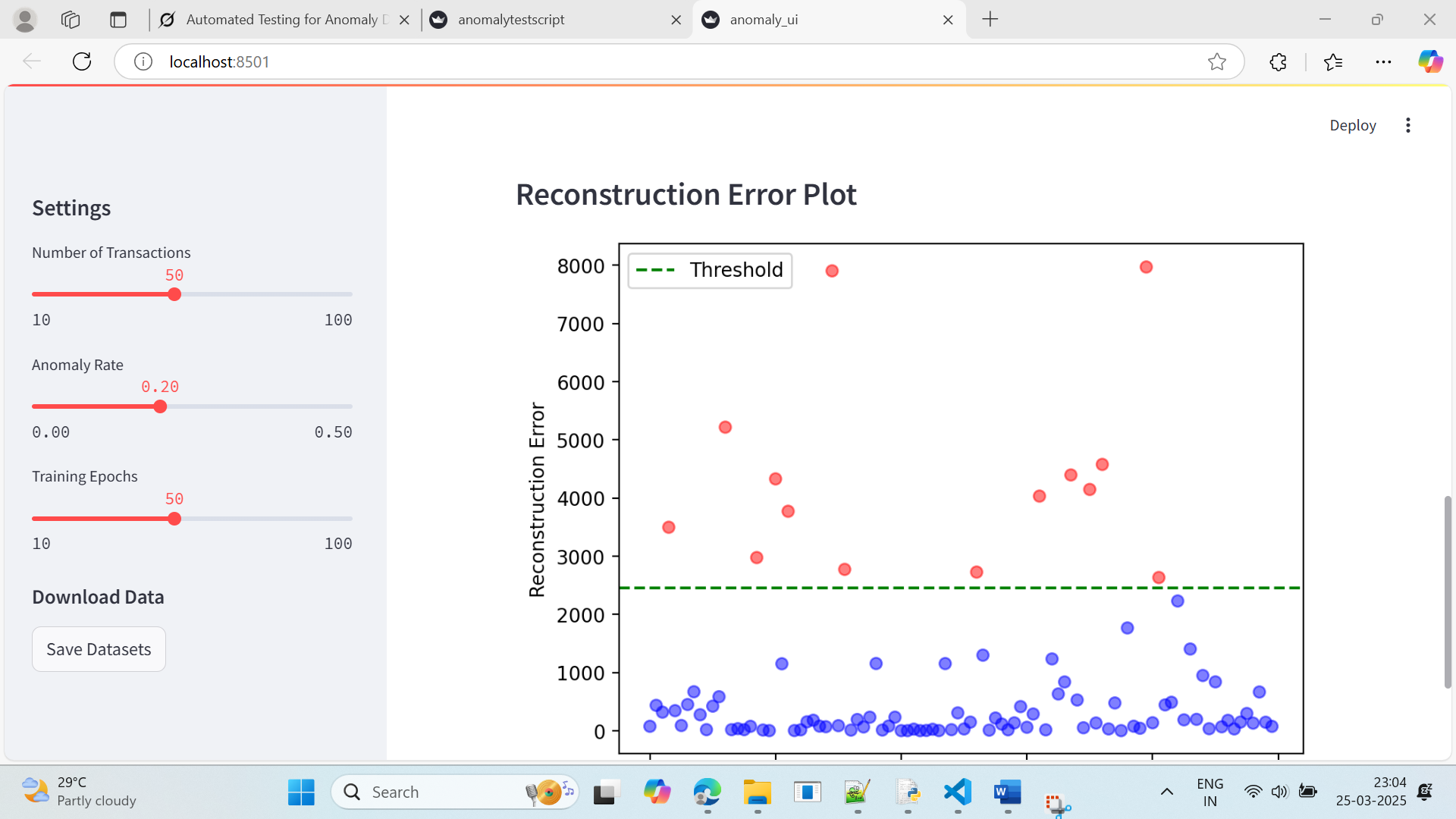
anomaly\_ui.py

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.

---

### What the Program Does

This program:

1. \*\*Makes Fake Bank Records\*\*: Creates pretend bank transactions to test the system.

2. \*\*Uses AI to Spot Weird Stuff\*\*: Trains an AI to figure out which transactions look odd.

3. \*\*Shows You the Results\*\*: Displays everything on a webpage where you can tweak settings, see tables, and look at a graph.

4. \*\*Lets You Save the Data\*\*: Gives you a way to download the fake bank records as files.

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

---

### The Big Pieces (Sections of the Code)

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

```python

import pandas as pd

import numpy as np

from datetime import datetime, timedelta

import torch

import torch.nn as nn

import torch.optim as optim

import streamlit as st

import matplotlib.pyplot as plt

```

- These are like the tools in a toolbox:

- `pandas`: A notebook to organize bank data (like a spreadsheet).

- `numpy`: A calculator for random numbers.

- `datetime` and `timedelta`: A calendar for dates.

- `torch`: The AI’s brainpower (a machine learning tool).

- `streamlit`: A magic wand to make a webpage.

- `matplotlib`: A sketchbook to draw graphs.

#### 2. \*\*Making Fake Bank Records (generate\_synthetic\_data)\*\*

```python

np.random.seed(42)

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

transaction\_ids = range(1, num\_records + 1)

dates = [base\_date + timedelta(days=i) for i in range(num\_records)]

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

num\_anomalies = int(num\_records \* anomaly\_rate)

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

amounts[anomaly\_indices] = np.random.uniform(5000, 10000, num\_anomalies)

data = pd.DataFrame({

'Transaction\_ID': transaction\_ids,

'Date': dates,

'Amount': amounts

})

return data

```

- \*\*What it does\*\*: Creates pretend bank transactions.

- \*\*How\*\*:

- `np.random.seed(42)`: Locks the random numbers so they’re the same every time (like setting a game level).

- `num\_records`: How many transactions (e.g., 50).

- `base\_date`: Starting date (e.g., March 1, 2025).

- Makes IDs (1, 2, 3, ...).

- Sets dates (March 1, March 2, ...).

- Picks amounts around $500, but keeps them between $50 and $1000.

- Adds some big, weird amounts ($5000-$10,000) for 20% of them (anomaly\_rate=0.2).

- Puts it in a table and gives it back.

#### 3. \*\*Building the AI Detective (VAE Class)\*\*

```python

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 is\*\*: A tiny AI brain called a Variational Autoencoder (VAE).

- \*\*How it works\*\*:

- Takes transaction amounts and squishes them into a smaller “summary.”

- Then rebuilds them to see if it got them right.

- `encoder`: Shrinks the data (like summarizing a book).

- `decoder`: Expands it back (like retelling the story).

- `reparameterize`: Adds a sprinkle of randomness to help the AI learn.

- `forward`: Runs the whole shrinking-and-rebuilding process.

#### 4. \*\*Grading the AI’s Work (vae\_loss)\*\*

```python

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

recon\_loss = nn.functional.mse\_loss(recon\_x, x, reduction='sum')

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

return recon\_loss + kl\_div

```

- \*\*What it does\*\*: Scores how well the AI rebuilt the transactions.

- \*\*How\*\*:

- `recon\_loss`: Checks if the rebuilt amounts match the originals (like a matching game).

- `kl\_div`: Keeps the AI honest so it doesn’t just memorize everything.

- Adds the two scores together to give the AI feedback.

#### 5. \*\*Training the AI Detective (train\_vae)\*\*

```python

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)

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 it does\*\*: Teaches the AI to spot weird transactions.

- \*\*How\*\*:

- Shrinks amounts to a 0-1 scale (like resizing pictures).

- Builds the VAE brain.

- Trains it `epochs` times (e.g., 50 practice rounds).

- Checks how far off the rebuilt amounts are (`recon\_error`).

- Sets a “weirdness” line (threshold) based on average error plus a bit extra.

- Flags anything above that line as odd.

- Gives back the odd flags, errors, and original min/max amounts.

#### 6. \*\*The Web Dashboard (main Function)\*\*

```python

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)

```

- \*\*What it does\*\*: Sets up a webpage to play with the AI.

- \*\*How\*\*:

- Puts a title at the top.

- Adds sliders on the side to pick:

- How many transactions (10 to 100, default 50).

- How many are weird (0% to 50%, default 20%).

- How long to train the AI (10 to 100 rounds, default 50).

```python

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)

```

- Makes two sets of fake data starting March 1, 2025.

- `data1`: Original fake transactions.

- `data2`: Copies `data1` but tweaks 20% of amounts slightly (80%-120% of original).

- Rounds amounts to 2 decimals (e.g., $500.23).

- Combines both sets into one big list for the AI.

```python

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

```

- Adds a “Detect Anomalies” button.

- When clicked, shows a “working…” spinner and:

- Trains the AI with the combined data.

- Adds error scores and “weird or not” flags to the table.

```python

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.")

```

- Shows:

- How many weird transactions were found.

- The “weirdness” line (threshold).

- A table of all transactions.

- A table of just the weird ones (or a message if none are found).

```python

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)

```

- Draws a graph:

- Dots for each transaction’s error (red = weird, blue = normal).

- A green line for the threshold.

- Labels and a legend to make it clear.

```python

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'")

```

- Adds a “Save Datasets” button on the side.

- When clicked, saves the two fake datasets as CSV files and says “Done!”

#### 7. \*\*Starting the Program\*\*

```python

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

main()

```

- Tells the program to start the webpage when you run it.

---

### How It All Fits Together

Imagine you’re a bank manager with a new AI assistant:

1. \*\*Make Fake Records\*\*: You create pretend transactions to test the AI.

2. \*\*Train the AI\*\*: The AI learns what’s normal and what’s weird.

3. \*\*Use the Dashboard\*\*: You open a webpage where you:

- Set how many transactions and how weird they are.

- Click “Detect Anomalies” to see what the AI finds.

- Look at tables and a graph to check the results.

- Save the data if you want.

4. \*\*See Results\*\*: The AI flags big spenders and shows you everything nicely.

---

### Running It

You run it with:

```powershell

& 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) where you can play with it.

---

### Why It’s Useful

This is like a practice run for a real bank system. It helps you test an AI detective without using real money, and the webpage makes it fun and easy to use. For a layman, think of it as a game where you’re the boss, and the AI is your helper finding sneaky transactions!

Let me know if you want me to explain any part more!

-----End of the Program description -----------

------------------------------------------------------------------------------------------------------

**Test script ran : (**Python unit test is created to check if the program is working fine)

Using AI written a program to test.

**anomalytestscript.py**

**What the Program Does**

The program tests four parts of a bigger system that uses artificial intelligence (AI) to spot weird bank transactions (like someone spending $10,000 when they usually spend $500). These tests make sure:

1. Fake bank data is created correctly.
2. The AI brain (called a "VAE") is built right.
3. The AI can learn and find odd transactions.
4. The system can say "nothing weird here" when there’s nothing unusual.

If all tests pass, it says "4 passed" instead of just "OK" to show everything worked.

**The Big Pieces (Sections of the Code)**

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

python

CollapseWrapCopy

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

* Think of these as our toolbox. We’re grabbing:
  + unittest: A checklist maker to test things.
  + pandas: A notebook to organize bank data (like a spreadsheet).
  + numpy: A calculator for numbers.
  + datetime and timedelta: A calendar to set dates.
  + torch: The AI’s brainpower (a library for machine learning).
  + io and sys: Tools to capture what the program says.
  + patch: A trick to pretend we’re printing messages.

**2. Making Fake Bank Data (generate\_synthetic\_data)**

python

CollapseWrapCopy

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

transaction\_ids = range(1, num\_records + 1)

dates = [base\_date + timedelta(days=i) for i in range(num\_records)]

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

num\_anomalies = int(num\_records \* anomaly\_rate)

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

amounts[anomaly\_indices] = np.random.uniform(5000, 10000, num\_anomalies)

data = pd.DataFrame({

'Transaction\_ID': transaction\_ids,

'Date': dates,

'Amount': amounts

})

return data

* **What it does**: Creates pretend bank records.
* **How**:
  + num\_records: How many transactions (e.g., 50).
  + base\_date: Starting date (e.g., March 1, 2025).
  + Makes IDs (1, 2, 3, ...).
  + Sets dates (March 1, March 2, ...).
  + Picks random amounts around $500 (most between $50 and $1000).
  + Adds some big, weird amounts (e.g., $5000-$10,000) for 20% of them (anomaly\_rate=0.2).
  + Puts it all in a table and hands it back.

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

python

CollapseWrapCopy

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 is**: A mini AI brain called a Variational Autoencoder (VAE).
* **How it works**:
  + Takes numbers (like transaction amounts) and squeezes them into a smaller form (like summarizing a story).
  + Then rebuilds them to check if it got the story right.
  + encoder: Shrinks the data.
  + decoder: Expands it back.
  + reparameterize: Adds a bit of randomness so the AI learns better.
  + forward: Runs the whole process.

**4. Scoring the AI’s Work (vae\_loss)**

python

CollapseWrapCopy

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

recon\_loss = nn.functional.mse\_loss(recon\_x, x, reduction='sum')

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

return recon\_loss + kl\_div

* **What it does**: Grades how well the AI rebuilt the data.
* **How**:
  + recon\_loss: Checks if the rebuilt data matches the original (like a spelling test).
  + kl\_div: Makes sure the AI doesn’t cheat by memorizing everything.
  + Adds the two scores together.

**5. Training the AI (train\_vae)**

python

CollapseWrapCopy

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 it does**: Teaches the AI to spot weird transactions.
* **How**:
  + Squishes amounts to a 0-1 scale (like resizing photos).
  + Creates the VAE brain.
  + Trains it epochs times (e.g., 50 practice rounds).
  + Checks how far off the rebuilt data is (recon\_error).
  + Sets a line (threshold) to say what’s weird (e.g., average error + some extra).
  + Flags anything above that line as an anomaly.

**6. Custom Test Report (CustomTestResult & CustomTestRunner)**

python

CollapseWrapCopy

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 it does**: Changes the test report to say "4 passed" instead of "OK".
* **How**:
  + CustomTestResult: Counts tests and checks if they all passed.
  + print\_result: Prints "4 passed" if no failures, or a failure message if something went wrong.
  + CustomTestRunner: Uses our custom report style.

**7. The Tests (TestAnomalyDetection)**

python

CollapseWrapCopy

class TestAnomalyDetection(unittest.TestCase):

* **Setup**: Prepares things like the date and number of records.
* **Four Tests**:
  1. **test\_generate\_synthetic\_data**:
     + Makes fake data and checks if it has the right number of rows, columns, IDs, dates, and some big amounts.
  2. **test\_vae\_model**:
     + Builds the AI brain and tests if it processes a number correctly (sizes match up).
  3. **test\_train\_vae**:
     + Trains the AI with fake data and checks if it finds anomalies and returns the right stuff.
  4. **test\_no\_anomalies\_output**:
     + Uses normal data (no big amounts), trains the AI, and checks if it says "No anomalies detected."

**8. Running the Tests**

python

CollapseWrapCopy

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

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

CustomTestRunner(verbosity=2).run(suite)

* **What it does**: Starts the tests and shows the results.
* **How**: Loads all four tests and uses our custom report to say "4 passed" if they work.

**How It All Fits Together**

Imagine you’re testing a toy robot that spots broken toys:

1. **Make fake toys**: Some normal, some broken (generate\_synthetic\_data).
2. **Build the robot’s brain**: Make sure it can think (VAE).
3. **Teach the robot**: Show it toys and let it learn what’s normal (train\_vae).
4. **Test it**:
   * Does it make toys right? (test\_generate\_synthetic\_data)
   * Does its brain work? (test\_vae\_model)
   * Can it spot broken toys? (test\_train\_vae)
   * Can it say "all good" when toys are fine? (test\_no\_anomalies\_output)
5. **Report**: If all tests pass, say "4 passed" (CustomTestRunner).

**Running It**

You run it with:

powershell

CollapseWrapCopy

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

* It’ll show each test running and end with "4 passed" if everything works.

**Why It’s Useful**

This is like a safety net for a bigger program. If someone changes the code later, these tests check if it still works. For a layman, think of it as a teacher grading a student’s homework—making sure every answer is right before saying "good job!"

Let me know if you want me to zoom in on any part!

Output is displayed in below screenshot.

