```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
from numpy.linalg import inv
from numpy import random
import os
```

```
!apt-get install openjdk-11-jdk-headless -qq
!pip install pyspark==3.4.4
!pip install graphframes
```

```
Requirement already satisfied: pyspark==3.4.4 in /usr/local/lib/python3.12/dist-packages (3.4.4)
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.12/dist-packages (from pyspark==3.4.4) (0.10.9.7)
Requirement already satisfied: graphframes in /usr/local/lib/python3.12/dist-packages (0.6)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from graphframes) (2.0.2)
Requirement already satisfied: nose in /usr/local/lib/python3.12/dist-packages (from graphframes) (1.3.7)
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
os.chdir(r"/content/drive/MyDrive/CDAC Project Final/Datsets")
```

Start Spark Session

Purpose: Initialize Spark so we can handle large transaction data.

```
from pyspark.sql import SparkSession
import os

# Explicitly set JAVA_HOME. This is a common fix for 'Java gateway process exited' in Colab.
# Verify the path to your Java installation. This is a typical path for openjdk-11 in Colab.
java_path = "/usr/lib/jvm/java-11-openjdk-amd64"
if os.path.exists(java_path):
    os.environ["JAVA_HOME"] = java_path
    print(f"JAVA_HOME set to: {os.environ['JAVA_HOME']}")
else:
    print(f"Warning: Java path not found at {java_path}. Please verify Java installation.")

spark = SparkSession.builder \
    .appName("AML-Graph") \
    .config("spark.jars.packages", "graphframes:graphframes:0.8.0-spark3.0-s_2.12") \
    .config("spark.sql.shuffle.partitions", "4") \
    .config("spark.driver.memory", "4g") \
    .getOrCreate()

print("Spark with GraphFrames started")
```

```
JAVA_HOME set to: /usr/lib/jvm/java-11-openjdk-amd64
Spark with GraphFrames started
```

VERIFY GRAPHFRAMES WORKS

```
from graphframes import GraphFrame

print("GraphFrames imported successfully")
```

```
GraphFrames imported successfully
```

Load HI-Small Transaction File

```
from pyspark.sql.functions import col

trans = spark.read.csv(
    "/content/drive/MyDrive/CDAC Project Final/Datsets/HI-Small_Trans.csv",
    header=True,
    inferSchema=True
)
```

```
trans.show(5)
```

```
+----------------+---------+---------+-------+---------+---------------+------------------+-----------+----------------+--------
|       Timestamp|From Bank| Account2|To Bank| Account4|Amount Received|Receiving Currency|Amount Paid|Payment Currency|Payment 
+----------------+---------+---------+-------+---------+---------------+------------------+-----------+----------------+--------
|2022/09/01 00:20|       10|8000EBD30|     10|8000EBD30|        3697.34|         US Dollar|    3697.34|       US Dollar|  Reinve
|2022/09/01 00:20|     3208|8000F4580|      1|8000F5340|           0.01|         US Dollar|       0.01|       US Dollar|        
|2022/09/01 00:00|     3209|8000F4670|   3209|8000F4670|       14675.57|         US Dollar|   14675.57|       US Dollar|  Reinve
|2022/09/01 00:02|       12|8000F5030|     12|8000F5030|        2806.97|         US Dollar|    2806.97|       US Dollar|  Reinve
|2022/09/01 00:06|       10|8000F5200|     10|8000F5200|       36682.97|         US Dollar|   36682.97|       US Dollar|  Reinve
+----------------+---------+---------+-------+---------+---------------+------------------+-----------+----------------+--------
only showing top 5 rows
```

```
trans.printSchema()
```

```
root
 |-- Timestamp: string (nullable = true)
 |-- From Bank: integer (nullable = true)
 |-- Account2: string (nullable = true)
 |-- To Bank: integer (nullable = true)
 |-- Account4: string (nullable = true)
 |-- Amount Received: double (nullable = true)
 |-- Receiving Currency: string (nullable = true)
 |-- Amount Paid: double (nullable = true)
 |-- Payment Currency: string (nullable = true)
 |-- Payment Format: string (nullable = true)
 |-- Is Laundering: integer (nullable = true)
```

```
print(f"Number of rows in 'trans' DataFrame: {trans.count()}")
print("\nDistinct values and their counts for 'Payment Format' (interpreted as transaction code):")
trans.groupBy("Payment Format").count().show()
```

```
Number of rows in 'trans' DataFrame: 5078345

Distinct values and their counts for 'Payment Format' (interpreted as transaction code):
+--------------+-------+
|Payment Format|  count|
+--------------+-------+
|       Bitcoin| 146091|
|  Reinvestment| 481056|
|   Credit Card|1323324|
|           ACH| 600797|
|        Cheque|1864331|
|          Cash| 490891|
|          Wire| 171855|
+--------------+-------+
```

Clean and Rename Columns

We now standardize column names for graph processing

```
from pyspark.sql.functions import col

transactions = trans.select(
    col("Account2").alias("src"),
    col("Account4").alias("dst"),
    col("Amount Paid").alias("amount"),
    col("Timestamp").alias("timestamp")
)
```

```
transactions.show(5)
```

```
+---------+---------+--------+----------------+
|      src|      dst|  amount|       timestamp|
+---------+---------+--------+----------------+
|8000EBD30|8000EBD30| 3697.34|2022/09/01 00:20|
|8000F4580|8000F5340|    0.01|2022/09/01 00:20|
|8000F4670|8000F4670|14675.57|2022/09/01 00:00|
|8000F5030|8000F5030| 2806.97|2022/09/01 00:02|
|8000F5200|8000F5200|36682.97|2022/09/01 00:06|
+---------+---------+--------+----------------+
only showing top 5 rows
```

STEP 2 — BUILDING THE TRANSACTION GRAPH (CORE LOGIC)

This step converts raw transaction rows into a graph structure, which is the foundation for detecting money laundering patterns.

Create Nodes (Vertices)

Each bank account becomes a node in the graph.

```
from pyspark.sql.functions import col

# Extract all unique accounts from sender and receiver
vertices = (
    transactions.select(col("src").alias("id"))
    .union(transactions.select(col("dst").alias("id")))
    .distinct()
)

vertices.show(5)
```

```
+---------+
|       id|
+---------+
|8000F5200|
|8000F5AD0|
|8005F2D30|
|8005FB700|
|8006A3840|
+---------+
only showing top 5 rows
```

Create Edges (Money Transfers)

Edges represent flow of money between accounts.

```
edges = transactions.select(
    col("src"),
    col("dst"),
    col("amount"),
    col("timestamp")
)
```

Build the GraphFrame

Now we combine vertices + edges.

```
from graphframes import GraphFrame

graph = GraphFrame(vertices, edges)
```

Validate the Graph

Always check before moving forward.

```
graph.vertices.show(5)
graph.edges.show(5)
```

```
+---------+
|       id|
+---------+
|8000F5200|
|8000F5AD0|
|8005F2D30|
|8005FB700|
|8006A3840|
+---------+
only showing top 5 rows

+---------+---------+--------+----------------+
|      src|      dst|  amount|       timestamp|
+---------+---------+--------+----------------+
|8000EBD30|8000EBD30| 3697.34|2022/09/01 00:20|
|8000F4580|8000F5340|    0.01|2022/09/01 00:20|
|8000F4670|8000F4670|14675.57|2022/09/01 00:00|
|8000F5030|8000F5030| 2806.97|2022/09/01 00:02|
|8000F5200|8000F5200|36682.97|2022/09/01 00:06|
+---------+---------+--------+----------------+
only showing top 5 rows
```

STEP 3 — DETECT SUSPICIOUS TRANSACTION PATTERNS (MOTIFS)

We will detect three classic money laundering behaviors:

1. Fan-Out – One account sending money to many accounts
2. Fan-In – Many accounts sending money to one account
3. Circular Transactions – Money moving in loops

```
Start coding or generate with AI.
```

FAN-OUT DETECTION (Sender ⟶ Many Receivers) Meaning:

One account distributing money to many others (possible layering or mule network).

```
fan_out = graph.outDegrees.orderBy("outDegree", ascending=False)
fan_out.show(10)
```

```
+---------+---------+
|       id|outDegree|
+---------+---------+
|100428660|   168672|
|1004286A8|   103018|
|100428978|    20497|
|1004286F0|    18663|
|100428780|    17264|
|1004289C0|    16794|
|100428810|    16426|
|1004287C8|    14174|
|100428738|    13756|
|100428A51|    13073|
+---------+---------+
only showing top 10 rows
```

FAN-IN DETECTION (Many ⟶ One)

Meaning:

Multiple accounts sending money to one account (aggregation point).

```
fan_in = graph.inDegrees.orderBy("inDegree", ascending=False)
fan_in.show(10)
```

```
+---------+--------+
|       id|inDegree|
+---------+--------+
|100428660|    1084|
|1004286A8|     653|
|80F47A310|     159|
|100428978|     150|
|8018859B0|     144|
|1004289C0|     132|
|100428780|     117|
|100428810|     114|
|80F0EF460|     109|
|1004286F0|     108|
+---------+--------+
only showing top 10 rows
```

```
Start coding or generate with AI.
```

CIRCULAR TRANSACTIONS (Layering)

Meaning:

Money flows in loops to hide origin.

```
cycles = graph.find("(a)-[e1]->(b); (b)-[e2]->(a)")
cycles.show(10)
```

```
+-----------+--------------------+-----------+--------------------+
|          a|                  e1|          b|                  e2|
+-----------+--------------------+-----------+--------------------+
|{800044A00}|{800044A00, 80004...|{800044A00}|{800044A00, 80004...|
|{800044A00}|{800044A00, 80004...|{800044A00}|{800044A00, 80004...|
|{800044A00}|{800044A00, 80004...|{800044A00}|{800044A00, 80004...|
|{800044A00}|{800044A00, 80004...|{800044A00}|{800044A00, 80004...|
|{800044A00}|{800044A00, 80004...|{800044A00}|{800044A00, 80004...|
|{800044A00}|{800044A00, 80004...|{800044A00}|{800044A00, 80004...|
|{800044A00}|{800044A00, 80004...|{800044A00}|{800044A00, 80004...|
|{800044A00}|{800044A00, 80004...|{800044A00}|{800044A00, 80004...|
|{800044A00}|{800044A00, 80004...|{800044A00}|{800044A00, 80004...|
|{800044A00}|{800044A00, 80004...|{800044A00}|{800044A00, 80004...|
+-----------+--------------------+-----------+--------------------+
only showing top 10 rows
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

STEP 4 — RISK SCORING (CORE INTELLIGENCE LAYER)

Combine Graph Metrics

We already computed:

fan_out (outDegree)

fan_in (inDegree)

Now we combine them into a single risk table.

```
from pyspark.sql.functions import col, when
```

```
# Join fan-in and fan-out scores
risk_df = fan_out.join(
    fan_in,
    on="id",
    how="outer"
).fillna(0)
```

Create a Risk Score Formula

```
risk_df = risk_df.withColumn(
    "risk_score",
    (col("outDegree") * 0.6) + (col("inDegree") * 0.4)
)
```

** Rank Accounts by Risk**

```
risk_df = risk_df.orderBy(col("risk_score").desc())
risk_df.show(10)
```

```
+---------+---------+--------+------------------+
|       id|outDegree|inDegree|        risk_score|
+---------+---------+--------+------------------+
|100428660|   168672|    1084|          101636.8|
|1004286A8|   103018|     653| 62071.99999999999|
|100428978|    20497|     150|12358.199999999999|
|1004286F0|    18663|     108|           11241.0|
|100428780|    17264|     117|10405.199999999999|
|1004289C0|    16794|     132|10129.199999999999|
|100428810|    16426|     114|            9901.2|
|1004287C8|    14174|     103|            8545.6|
|100428738|    13756|      98| 8292.800000000001|
|100428A51|    13073|      28| 7854.999999999999|
+---------+---------+--------+------------------+
only showing top 10 rows
```

OPTIONAL: Add Risk Label

To make results more interpretable:

```
risk_df = risk_df.withColumn(
    "risk_label",
    when(col("risk_score") >= 10, "HIGH")
    .when(col("risk_score") >= 5, "MEDIUM")
    .otherwise("LOW")
)
```

STEP 4.1 — Verify Risk Scores Before Moving Ahead

Display Top Risky Accounts

```
risk_df.show(10)
```

```
+---------+---------+--------+------------------+----------+
|       id|outDegree|inDegree|        risk_score|risk_label|
+---------+---------+--------+------------------+----------+
|100428660|   168672|    1084|          101636.8|      HIGH|
|1004286A8|   103018|     653| 62071.99999999999|      HIGH|
|100428978|    20497|     150|12358.199999999999|      HIGH|
|1004286F0|    18663|     108|           11241.0|      HIGH|
|100428780|    17264|     117|10405.199999999999|      HIGH|
|1004289C0|    16794|     132|10129.199999999999|      HIGH|
|100428810|    16426|     114|            9901.2|      HIGH|
|1004287C8|    14174|     103|            8545.6|      HIGH|
|100428738|    13756|      98| 8292.800000000001|      HIGH|
|100428A51|    13073|      28| 7854.999999999999|      HIGH|
+---------+---------+--------+------------------+----------+
only showing top 10 rows
```

Check Distribution (Sanity Check)

We want to see if only few accounts are risky (which is realistic).

```
risk_df.select("risk_score").describe().show()
```

```
+-------+------------------+
|summary|        risk_score|
+-------+------------------+
|  count|            515080|
|   mean| 9.859332530871262|
| stddev|171.99636112558056|
|    min|               0.4|
|    max|          101636.8|
+-------+------------------+
```

STEP 4.2 — LABEL HIGH-RISK ACCOUNTS (THRESHOLDING)

Apply Risk Labels in Spark

```
from pyspark.sql.functions import when

risk_labeled = risk_df.withColumn(
    "risk_label",
    when(col("risk_score") >= 8, "HIGH")
    .when(col("risk_score") >= 4, "MEDIUM")
    .otherwise("LOW")
)
```

View Final Risk Table

```
risk_labeled.orderBy(col("risk_score").desc()).show(10)
```

```
+---------+---------+--------+------------------+----------+
|       id|outDegree|inDegree|        risk_score|risk_label|
+---------+---------+--------+------------------+----------+
|100428660|   168672|    1084|          101636.8|      HIGH|
|1004286A8|   103018|     653| 62071.99999999999|      HIGH|
|100428978|    20497|     150|12358.199999999999|      HIGH|
|1004286F0|    18663|     108|           11241.0|      HIGH|
|100428780|    17264|     117|10405.199999999999|      HIGH|
|1004289C0|    16794|     132|10129.199999999999|      HIGH|
|100428810|    16426|     114|            9901.2|      HIGH|
|1004287C8|    14174|     103|            8545.6|      HIGH|
|100428738|    13756|      98| 8292.800000000001|      HIGH|
|100428A51|    13073|      28| 7854.999999999999|      HIGH|
+---------+---------+--------+------------------+----------+
only showing top 10 rows
```

```
Start coding or generate with AI.
```

STEP 5 — VISUALIZE HIGH-RISK ACCOUNTS IN THE TRANSACTION GRAPH

Convert Spark Graph to NetworkX

```
import networkx as nx
import matplotlib.pyplot as plt
```

Convert edges to pandas first:

```
edges_pd = graph.edges.select("src", "dst").toPandas()
```

Create NetworkX graph:

```
G = nx.from_pandas_edgelist(edges_pd, source="src", target="dst", create_using=nx.DiGraph())
```

STEP 5.2 — Attach Risk Scores to Nodes

We need to color nodes by risk level.

```
risk_pd = risk_labeled.select("id", "risk_label").toPandas()

risk_map = dict(zip(risk_pd["id"], risk_pd["risk_label"]))
```

Assign colors:

```
node_colors = []
for node in G.nodes():
    if node in risk_map:
        if risk_map[node] == "HIGH":
            node_colors.append("red")
        elif risk_map[node] == "MEDIUM":
            node_colors.append("orange")
        else:
            node_colors.append("green")
    else:
        node_colors.append("gray")
```

```
Start coding or generate with AI.
```

FILTER ONLY HIGH-RISK ACCOUNTS

```
# Get top risky nodes
high_risk_nodes = risk_labeled \
    .filter(col("risk_label") == "HIGH") \
    .select("id") \
    .limit(10) \
    .rdd.flatMap(lambda x: x) \
    .collect()
```

Create Subgraph (Safe & Fast)

```
sub_edges = edges_pd[
    (edges_pd["src"].isin(high_risk_nodes)) |
    (edges_pd["dst"].isin(high_risk_nodes))
]

G_sub = nx.from_pandas_edgelist(
    sub_edges,
    source="src",
    target="dst",
    create_using=nx.DiGraph()
)
```

```
print("Number of nodes:", G_sub.number_of_nodes())
print("Number of edges:", G_sub.number_of_edges())
```

```
Number of nodes: 35458
Number of edges: 35449
```

SAFE VISUALIZATION (TOP-N NODES ONLY)

```
# Take top 20 nodes only
top_nodes = list(G_sub.nodes())[:20]
G_small = G_sub.subgraph(top_nodes)
```

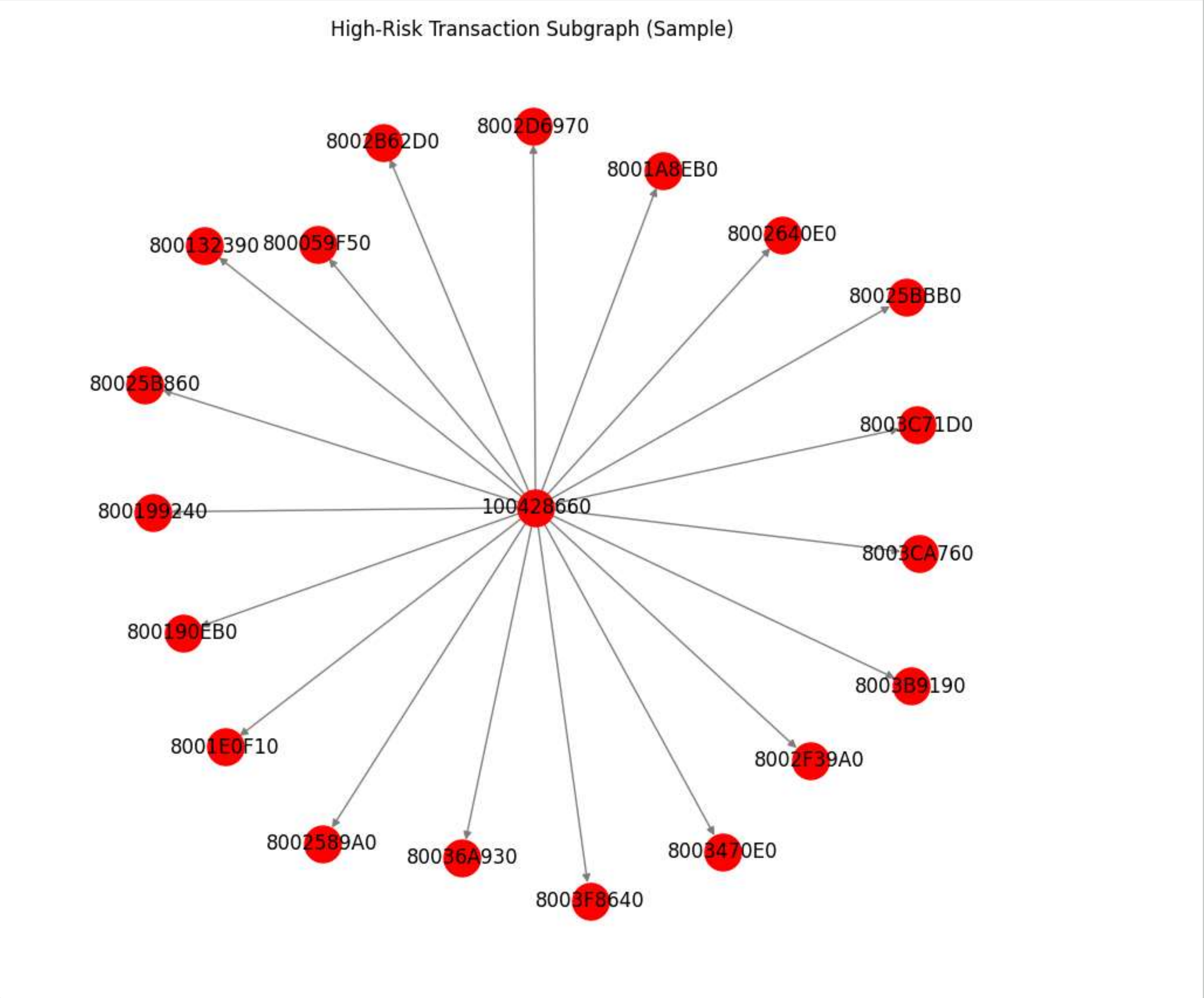
```
plt.figure(figsize=(8, 8))

pos = nx.spring_layout(G_small, seed=42, k=0.5)

nx.draw(
    G_small,
    pos,
    with_labels=True,
    node_color="red",
    node_size=500,
    edge_color="gray"
)

plt.title("High-Risk Transaction Subgraph (Sample)")
```

```
    plt.show()
```



```
    Start coding or generate with AI.
```

STEP 6 — INTRODUCTION TO GNN (Graph Neural Network)

STEP 6.1 — PREPARE GRAPH FOR GNN

STEP 6.1.1 — Convert Spark Graph → Pandas

```
# Convert edges to pandas
edges_pd = edges.select("src", "dst").toPandas()
```

STEP 6.1.2 — Create Node Index Mapping

```
import pandas as pd

nodes = pd.unique(edges_pd[['src', 'dst']].values.ravel())
node_map = {node: i for i, node in enumerate(nodes)}
```

STEP 6.1.3 — Build Edge Index Tensor

```
import torch

edge_index = torch.tensor([
    [node_map[src] for src in edges_pd['src']],
    [node_map[dst] for dst in edges_pd['dst']]
], dtype=torch.long)
```

STEP 6.1.4 — Create Node Features

For now, we use simple features:

Degree-based features (safe + effective)

```
import numpy as np

num_nodes = len(node_map)
x = torch.zeros((num_nodes, 1))  # simple 1D feature

# Optional: degree-based feature
for src, dst in zip(edges_pd['src'], edges_pd['dst']):
    x[node_map[src]] += 1
    x[node_map[dst]] += 1
```

STEP 6.1.5 — Build PyG Data Object

```
!pip install torch_geometric
from torch_geometric.data import Data

data = Data(
    x=x,
    edge_index=edge_index
)
```

```
Collecting torch_geometric
  Downloading torch_geometric-2.7.0-py3-none-any.whl.metadata (63 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 63.7/63.7 kB 3.2 MB/s eta 0:00:00
Requirement already satisfied: aiohttp in /usr/local/lib/python3.12/dist-packages (from torch_geometric) (3.13.2)
Requirement already satisfied: fsspec in /usr/local/lib/python3.12/dist-packages (from torch_geometric) (2025.3.0)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from torch_geometric) (3.1.6)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from torch_geometric) (2.0.2)
Requirement already satisfied: psutil>=5.8.0 in /usr/local/lib/python3.12/dist-packages (from torch_geometric) (5.9.5)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.12/dist-packages (from torch_geometric) (3.2.5)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from torch_geometric) (2.32.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from torch_geometric) (4.67.1)
Requirement already satisfied: xxhash in /usr/local/lib/python3.12/dist-packages (from torch_geometric) (3.6.0)
Requirement already satisfied: aiohappyeyeballs>=2.5.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp->torch_geometric
Requirement already satisfied: aiosignal>=1.4.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp->torch_geometric) (1.4.
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp->torch_geometric) (25.4.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.12/dist-packages (from aiohttp->torch_geometric) (1.8
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.12/dist-packages (from aiohttp->torch_geometric) (6
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp->torch_geometric) (0.4.
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp->torch_geometric) (1.2
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2->torch_geometric) (3.0.3)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests->torch_geometr
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests->torch_geometric) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests->torch_geometric) (2
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests->torch_geometric) (2
Requirement already satisfied: typing-extensions>=4.2 in /usr/local/lib/python3.12/dist-packages (from aiosignal>=1.4.0->aiohttp
Downloading torch_geometric-2.7.0-py3-none-any.whl (1.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 32.1 MB/s eta 0:00:00
Installing collected packages: torch_geometric
Successfully installed torch_geometric-2.7.0
```

```
Start coding or generate with AI.
```

STEP 7 — TRAIN A GRAPH NEURAL NETWORK (GNN

STEP 7.1 — Define the GNN Model

We'll use a simple GCN (Graph Convolutional Network) — perfect for learning structural patterns.

```
import torch
import torch.nn.functional as F
from torch_geometric.nn import GCNConv
```

```
class AML_GCN(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = GCNConv(1, 16)
        self.conv2 = GCNConv(16, 1)

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = self.conv2(x, edge_index)
        return torch.sigmoid(x)
```

STEP 7.2 — Prepare Training Targets

Since we don't have real labels, we use weak supervision.

```
# create labels based on previous risk scoring
labels = torch.zeros((data.num_nodes, 1))

for i, node in enumerate(node_map):
    if node in high_risk_nodes:
        labels[i] = 1
```

STEP 7.3 — Train the Model

```
model = AML_GCN()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)

for epoch in range(50):
    model.train()
    optimizer.zero_grad()

    out = model(data)
    loss = F.binary_cross_entropy(out, labels)

    loss.backward()
    optimizer.step()

    if epoch % 10 == 0:
        print(f"Epoch {epoch}, Loss: {loss.item():.4f}")
```

```
Epoch 0, Loss: 20.0752
Epoch 10, Loss: 0.2818
Epoch 20, Loss: 0.2102
Epoch 30, Loss: 0.1745
Epoch 40, Loss: 0.1506
```

STEP 7.4 — Interpret Model Output

```
with torch.no_grad():
    predictions = model(data).squeeze()
```

```
top_risky = torch.argsort(predictions, descending=True)[:10]
top_risky
```

```
tensor([ 65256, 276765, 414729,  65237,  65242, 414741, 276685, 151007, 276660,
        414755])
```

```
Start coding or generate with AI.
```

STEP 8 — AUTOMATED SAR (Suspicious Activity Report) GENERATION

STEP 8.1 — Select High-Risk Accounts

We define a threshold (example: top 5%).

```
import numpy as np

# Convert risk_labeled Spark DataFrame to Pandas DataFrame for numpy.percentile
# Or, if you want to keep it in Spark, you'd use Spark SQL functions for percentile.
# For consistency with the numpy percentile usage, let's convert to pandas for this step.
risk_labeled_pd = risk_labeled.select("risk_score").toPandas()

threshold = np.percentile(risk_labeled_pd["risk_score"], 95)
high_risk_accounts_spark = risk_labeled.filter(risk_labeled["risk_score"] >= threshold)

high_risk_accounts_spark.show(5)
```

```
+---------+---------+--------+------------------+----------+
|       id|outDegree|inDegree|        risk_score|risk_label|
+---------+---------+--------+------------------+----------+
|100428660|   168672|    1084|          101636.8|      HIGH|
|1004286A8|   103018|     653| 62071.99999999999|      HIGH|
|100428978|    20497|     150|12358.199999999999|      HIGH|
|1004286F0|    18663|     108|           11241.0|      HIGH|
|100428780|    17264|     117|10405.199999999999|      HIGH|
+---------+---------+--------+------------------+----------+
only showing top 5 rows
```

STEP 8.2 — Create Natural Language Evidence

We create structured text describing suspicious behavior.

```
reports = []

# Convert the Spark DataFrame to a Pandas DataFrame for iteration
high_risk_accounts_pd = high_risk_accounts_spark.toPandas()

for _, row in high_risk_accounts_pd.iterrows():
    report = f"""
    Account {row['id']} has been flagged as HIGH RISK (Risk Score: {row['risk_score']:.2f}).
    The account exhibits unusual transactional behavior compared to peers (Out-Degree: {row['outDegree']}, In-Degree: {row['inDe
    The model detected abnormal connectivity patterns indicating potential money laundering.
    """
    reports.append(report)

print(f"Generated {len(reports)} SARs for high-risk accounts.")
for i, sar in enumerate(reports[:5]): # Print first 5 reports as an example
    print(f"--- SAR {i+1} ---")
    print(sar)
    print("-------------------")
```

```
Generated 25937 SARs for high-risk accounts.
--- SAR 1 ---

    Account 100428660 has been flagged as HIGH RISK (Risk Score: 101636.80).
    The account exhibits unusual transactional behavior compared to peers (Out-Degree: 168672, In-Degree: 1084).
    The model detected abnormal connectivity patterns indicating potential money laundering.

-------------------
--- SAR 2 ---

    Account 1004286A8 has been flagged as HIGH RISK (Risk Score: 62072.00).
    The account exhibits unusual transactional behavior compared to peers (Out-Degree: 103018, In-Degree: 653).
    The model detected abnormal connectivity patterns indicating potential money laundering.

-------------------
--- SAR 3 ---

    Account 100428978 has been flagged as HIGH RISK (Risk Score: 12358.20).
    The account exhibits unusual transactional behavior compared to peers (Out-Degree: 20497, In-Degree: 150).
    The model detected abnormal connectivity patterns indicating potential money laundering.

-------------------
--- SAR 4 ---
```