The code builds a small example of **classifying users as benign or malicious** using a **Graph Neural Network (GNN)** with the **GraphSAGE** model.
It uses a tiny fake graph to show how AI can learn patterns from connected data .

Section 1 — Installing & Importing

```
!pip install torch_geometric
import torch
from torch_geometric.data import Data
from torch_geometric.nn import SAGEConv
import torch.nn.functional as F
```

- The first line installs `torch_geometric`, which allows us to work with graph (network) data.
- `torch` is used for numbers and AI calculations.
- `Data` stores the graph.
- SAGEConv is the main **GraphSAGE neural network layer**.
- F helps apply math functions like activation (ReLU).

Section 2 — Creating Node Features

```
x = torch.tensor([
  [1.0, 0.0],
  [0.9, 0.1],
  [0.8, 0.2],
  [0.0, 1.0],
  [0.1, 0.9],
  [0.2, 0.8]
])
```

- x holds the **features of each node (user)**.

- The graph has **6 users**, and each gets 2 numbers to describe behavior.
- Higher values in first column ≈ normal user.
- Higher in second column ≈ suspicious user.

Section 3 — Defining Connections (Edges)

```
edge_index = torch.tensor([
  [0, 1, 2, 3, 4, 5],
  [1, 0, 1, 4, 3, 3]
], dtype=torch.long)
```

- `edge_index` describes who is connected to who.
- This makes a small **social network-like structure**.
- The graph is treated as **undirected** (connection works both ways).

Section 4 — Adding Labels

```
y = torch.tensor([0, 0, 1, 1, 1, 1], dtype=torch.long)
data = Data(x=x, edge_index=edge_index, y=y)
```

- y marks which users are benign (`0`) and malicious (`1`).
- We package the graph into a `Data` object so the AI model can learn from it.

Section 5 — Building the GraphSAGE Model

```
class Net(torch.nn.Module):
  def __init__(self):
    super(Net,self).__init__()
    self.conv1 = SAGEConv(2,4)
    self.conv2 = SAGEConv(4,2)

  def forward(self,data):
```

```
    x, e = data.x, data.edge_index
    x = self.conv1(x,e)
    x = F.relu(x)
    x = self.conv2(x,e)
    return x
```

- We create a model named `Net` that has:
    - a GraphSAGE layer (takes 2 numbers → outputs 4)
    - ReLU activation (adds non-linearity)
    - another GraphSAGE layer (4 → 2 for final classes)
- The model looks at connected users to decide if someone is malicious.

Section 6 — Training the AI

```
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
for i in range(20):
  optimizer.zero_grad()
  out = model(data)
  loss = F.cross_entropy(out, data.y)
  loss.backward()
  optimizer.step()
```

- We loop **20 times** to teach AI.
- `loss` measures how wrong the predictions are.
- We update (`step()`) until AI learns better.

Section 8 — Making Predictions

```
model.eval()
pred = out.argmax(dim=1)
print("Predicted labels:", pred.tolist())
```

- After learning, we check what AI predicts.

- The program prints final guessed labels (benign or malicious).