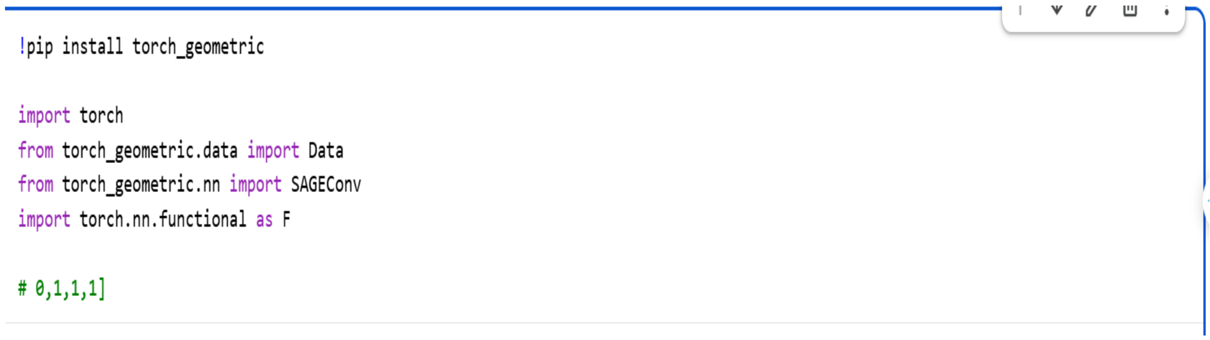
# Sara Nabil Tahawy 2205224 Report (Sec 9)

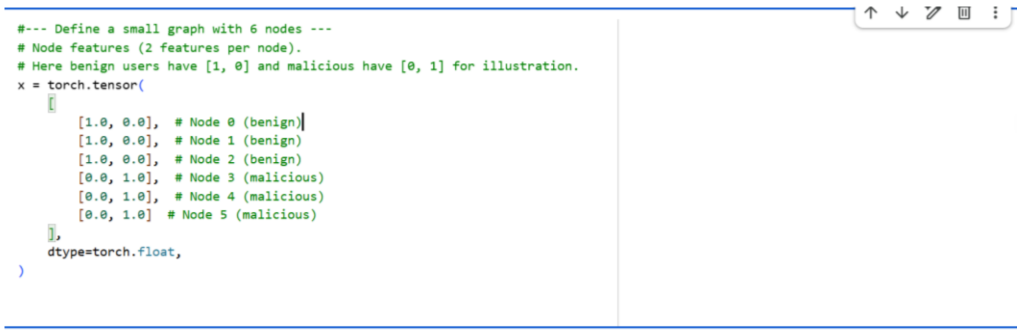


```
!pip install torch_geometric

import torch
from torch_geometric.data import Data
from torch_geometric.nn import SAGEConv
import torch.nn.functional as F

# 0,1,1,1]
```

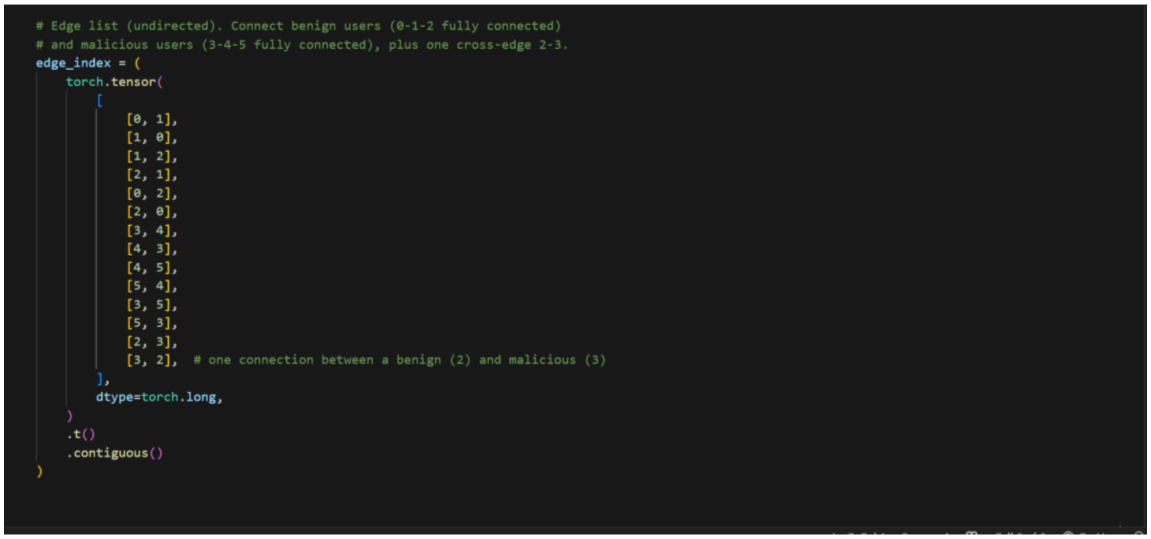
- !pip install torch_geometric : install the library to work with graphs.
- import torch : for numbers and arrays (tensors).
- from torch_geometric.data import Data : to create a graph with nodes and edges.
- from torch_geometric.nn import SAGEConv :a layer that learns from nodes and their neighbors.
- import torch.nn.functional as F : for functions like ReLU or softmax



```
#--- Define a small graph with 6 nodes ---
# Node features (2 features per node).
# Here benign users have [1, 0] and malicious have [0, 1] for illustration.
x = torch.tensor(
    [
        [1.0, 0.0],  # Node 0 (benign)
        [1.0, 0.0],  # Node 1 (benign)
        [1.0, 0.0],  # Node 2 (benign)
        [0.0, 1.0],  # Node 3 (malicious)
        [0.0, 1.0],  # Node 4 (malicious)
        [0.0, 1.0]  # Node 5 (malicious)
    ],
    dtype=torch.float,
)
```

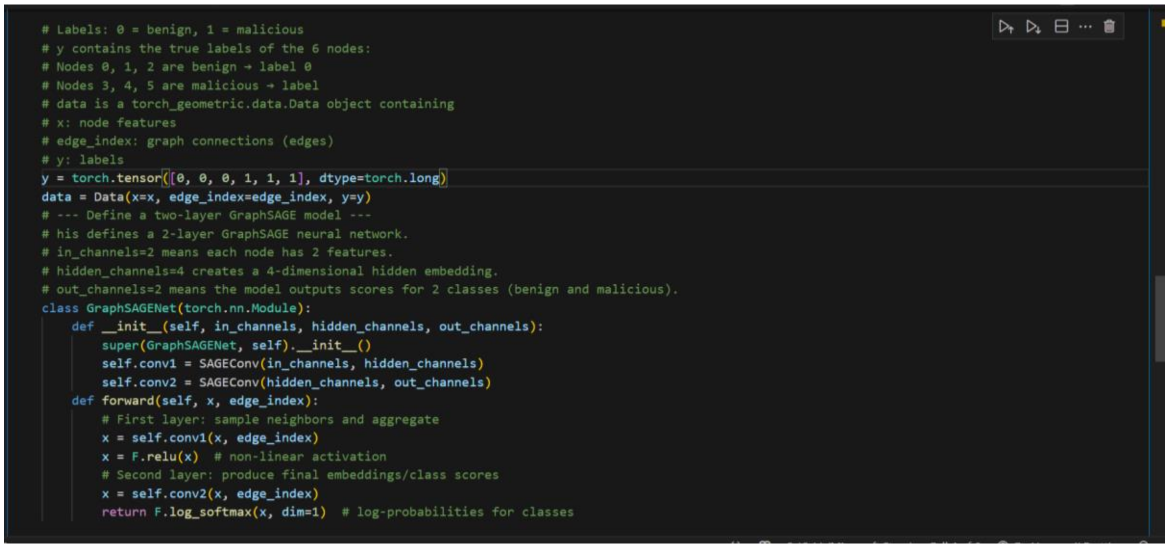
- Each node has **2 features**
- [1.0, 0.0] → benign node
- [0.0, 1.0] → malicious node

- This tensor x stores all node features for the graph.



```python
# Edge list (undirected). Connect benign users (0-1-2 fully connected)
# and malicious users (3-4-5 fully connected), plus one cross-edge 2-3.
edge_index = (
    torch.tensor(
        [
            [0, 1],
            [1, 0],
            [1, 2],
            [2, 1],
            [0, 2],
            [2, 0],
            [3, 4],
            [4, 3],
            [4, 5],
            [5, 4],
            [3, 5],
            [5, 3],
            [2, 3],
            [3, 2],  # one connection between a benign (2) and malicious (3)
        ],
        dtype=torch.long,
    )
    .t()
    .contiguous()
)
```

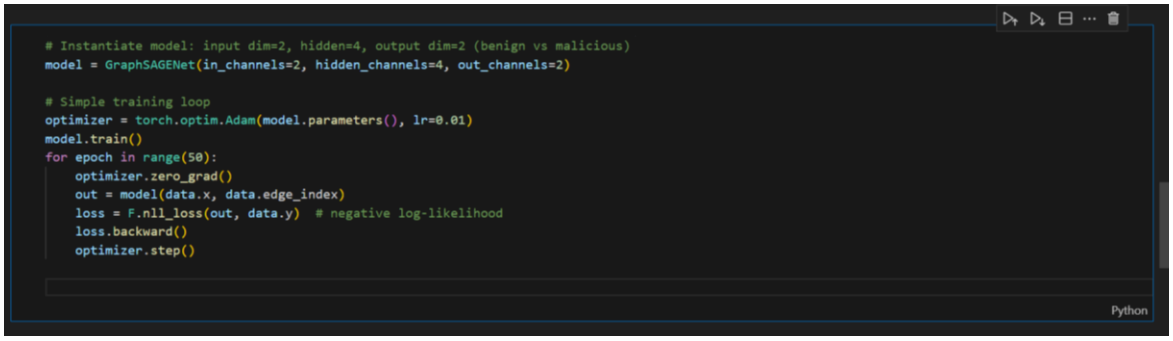
- **Note** : I take screen from VS Code instead of Colab to insert only one pic.
- **Nodes 0-1-2** (benign) are **fully connected**.
- **Nodes 3-4-5** (malicious) are **fully connected**.
- There is **one edge between a benign node and a malicious node**: node 2 connects to node 3



```python
# Labels: 0 = benign, 1 = malicious
# y contains the true labels of the 6 nodes:
# Nodes 0, 1, 2 are benign → label 0
# Nodes 3, 4, 5 are malicious → label
# data is a torch_geometric.data.Data object containing
# x: node features
# edge_index: graph connections (edges)
# y: labels
y = torch.tensor([0, 0, 0, 1, 1, 1], dtype=torch.long)
data = Data(x=x, edge_index=edge_index, y=y)
# --- Define a two-layer GraphSAGE model ---
# his defines a 2-layer GraphSAGE neural network.
# in_channels=2 means each node has 2 features.
# hidden_channels=4 creates a 4-dimensional hidden embedding.
# out_channels=2 means the model outputs scores for 2 classes (benign and malicious).
class GraphSAGENet(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels):
        super(GraphSAGENet, self).__init__()
        self.conv1 = SAGEConv(in_channels, hidden_channels)
        self.conv2 = SAGEConv(hidden_channels, out_channels)
    def forward(self, x, edge_index):
        # First layer: sample neighbors and aggregate
        x = self.conv1(x, edge_index)
        x = F.relu(x)  # non-linear activation
        # Second layer: produce final embeddings/class scores
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)  # log-probabilities for classes
```

- y : true labels for nodes: 0 = benign, 1 = malicious.

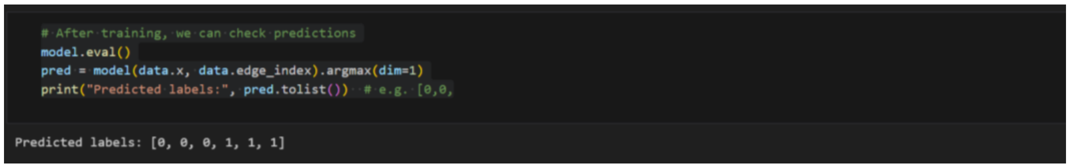
- data : contains node features (x), edges (edge_index), and labels (y).
- GraphSAGENet : a 2-layer GraphSAGE network:
    - **Layer 1**: looks at each node and its neighbors (conv1) + ReLU.
    - **Layer 2**: outputs scores for 2 classes (conv2) → log probabilities.



```python
# Instantiate model: input dim=2, hidden=4, output dim=2 (benign vs malicious)
model = GraphSAGENet(in_channels=2, hidden_channels=4, out_channels=2)

# Simple training loop
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
model.train()
for epoch in range(50):
    optimizer.zero_grad()
    out = model(data.x, data.edge_index)
    loss = F.nll_loss(out, data.y)  # negative log-likelihood
    loss.backward()
    optimizer.step()
```

- model = GraphSAGENet : make the model (2 input features, 4 hidden, 2 output).
- optimizer = Adam: helps the model learn.
- Loop 50 times:
- get predictions : out = model()
- calculate loss : F.nll_loss
- update model() : loss.backward() + optimizer.step



```python
# After training, we can check predictions
model.eval()
pred = model(data.x, data.edge_index).argmax(dim=1)
print("Predicted labels:", pred.tolist())  # e.g. [0,0,
```

```
Predicted labels: [0, 0, 0, 1, 1, 1]
```

## Check Predictions:

- model.eval() :stop training.
- pred = model(...).argmax(1) : predicted labels for nodes.
- Example output: [0, 0, 0, 1, 1, 1] : 0 = good, 1 = bad.

## Thank You