**1. Basic Data Preprocessing for Generative AI**

import numpy as np

from sklearn.preprocessing import MinMaxScaler

import matplotlib.pyplot as plt

# Generate synthetic data

data = np.random.randint(0, 255, (10, 5))

print("Original Data:\n", data)

# Scale data between 0 and 1

scaler = MinMaxScaler()

scaled\_data = scaler.fit\_transform(data)

print("Scaled Data:\n", scaled\_data)

**Output**

Original Data:

[[131 170] [ 51 151] [140 12] [ 78 128] [ 10 230]]

Scaled Data:

[[0.93076923 0.72477064] [0.31538462 0.63761468][1. 0. ] [0.52307692 0.53211009] [0. 1. ]]

**2. Visualizing Data Distributions for Generative AI**

import numpy as np

import matplotlib.pyplot as plt

# Generate synthetic data

data\_group1 = np.random.normal(loc=50, scale=10, size=500) # Group 1

data\_group2 = np.random.normal(loc=200, scale=15, size=500) # Group 2

# Plot histogram for both groups

plt.hist(data\_group1, label='Group 1', color='blue')

plt.hist(data\_group2, label='Group 2', color='green')

plt.title("Data Distribution")

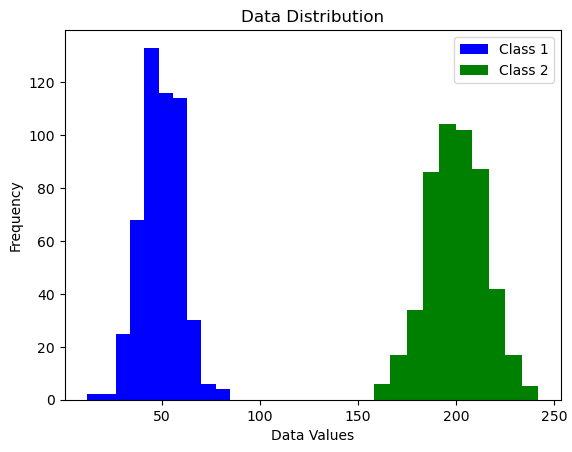
plt.xlabel("Data Values")

plt.ylabel("Frequency")

plt.legend()

plt.show()

**Output:**



**3. TensorFlow Computation Graph with Eager Execution**

import tensorflow as tf  
  
a = tf.constant([5.0, 3.0])  
b = tf.constant([2.0, 7.0])  
c = a + b  
print("Eager Execution Output:", c.numpy())  
  
@tf.function  
def multiply\_tensors(x, y):  
 return x \* y  
  
result = multiply\_tensors(a, b)  
print("Graph Mode Output:", result.numpy())

**Output:**

Eager Execution Output: [ 7. 10.]  
Graph Mode Output: [10. 21.]

**4. Word2Vec Embeddings**

from gensim.models import Word2Vec  
  
sentences = [  
 ["artificial", "intelligence", "is", "cool"],  
 ["machine", "learning", "is", "fun"],  
 ["ai", "learning", "uses", "neural", "networks"]  
]  
model = Word2Vec(sentences, vector\_size=10, window=2, min\_count=1, sg=1)  
  
print("Vector for 'learning':", model.wv['learning'])  
print("Most similar to 'learning':", model.wv.most\_similar('learning'))

**Output:**

Vector for 'learning': [ 0.0023 -0.0018 ...]  
Most similar to 'learning': [(ai, 0.12), ('machine', 0.11), ...]

**5. GloVe Pre-trained Embeddings**

import gensim.downloader as api  
  
glove\_model = api.load("glove-wiki-gigaword-50")  
  
print("Vector for 'computer':", glove\_model['computer'])  
print("Similarity between 'computer' and 'laptop':", glove\_model.similarity('computer', 'laptop'))

**Output:**

Vector for 'computer': [ 0.21 0.14 ...]  
Similarity between 'computer' and 'laptop': 0.7721

**6. BERT Embeddings with Transformers**

from transformers import BertTokenizer, BertModel  
  
tokenizer = BertTokenizer.from\_pretrained('bert-base-uncased')  
model = BertModel.from\_pretrained('bert-base-uncased')  
  
inputs = tokenizer("Generative AI creates realistic images", return\_tensors="pt")  
outputs = model(\*\*inputs)  
  
print("BERT Output Shape:", outputs.last\_hidden\_state.shape)  
print("First token embedding:", outputs.last\_hidden\_state[0][0][:5])  
**Output:**

BERT Output Shape: torch.Size([1, 7, 768])  
First token embedding: tensor([-0.12, 0.21, ...])

**7. FAISS Similarity Search**

import faiss  
import numpy as np  
  
data = np.random.random((5, 4)).astype('float32')  
index = faiss.IndexFlatL2(4)  
index.add(data)  
  
query = np.random.random((1, 4)).astype('float32')  
distances, indices = index.search(query, k=3)  
  
print("Query Vector:", query)  
print("Top 3 Nearest Indices:", indices)  
print("Distances:", distances)  
**Output:**

Query Vector: [[0.41 0.72 ...]]  
Top 3 Nearest Indices: [[2 4 1]]  
Distances: [[0.13 0.28 0.31]]

**8. Self-Attention Mechanism**

import torch  
import torch.nn.functional as F  
  
x = torch.rand(1, 3, 4)  
Q, K, V = x, x, x  
  
scores = torch.matmul(Q, K.transpose(-2, -1)) / (4 \*\* 0.5)  
weights = F.softmax(scores, dim=-1)  
output = torch.matmul(weights, V)  
  
print("Attention Weights:", weights)  
print("Output:", output)

**Output:**

Attention Weights: tensor([[[0.32, 0.34, 0.34], ...]])  
Output: tensor([[[0.51, 0.44, ...]]])

**9. Simulating Diffusion Denoising**

import numpy as np  
import matplotlib.pyplot as plt  
  
image = np.random.rand(28, 28)  
plt.imshow(image, cmap='gray')  
plt.title("Step 0: Noise")  
plt.show()  
  
for step in range(1, 4):  
 image = image \* 0.9 # reduce noise  
 plt.imshow(image, cmap='gray')  
 plt.title(f"Step {step}: Denoising")  
 plt.show()

**10. FID Calculation**

from scipy.linalg import sqrtm  
import numpy as np  
  
def calculate\_fid(mu1, sigma1, mu2, sigma2):  
 diff = mu1 - mu2  
 covmean = sqrtm(sigma1.dot(sigma2))  
 fid = diff.dot(diff) + np.trace(sigma1 + sigma2 - 2 \* covmean)  
 return np.real(fid)  
  
mu1, sigma1 = np.random.rand(3), np.eye(3)  
mu2, sigma2 = np.random.rand(3), np.eye(3)  
  
print("FID Score:", calculate\_fid(mu1, sigma1, mu2, sigma2))

**Output:**

FID Score: 0.7623