Lab Assignment 3:

Objective: To imeplement differnt distance metrics in ML.

Name: AAKASH VERMA

Reg. No.: 24-08-26

Course: M.Tech. (Cyber Security)

In [2]:
```
import numpy as np
```

In [3]:
```
def euclideanDistance(x, y):
    return np.sqrt(np.sum((np.array(x) - np.array(y))**2))

# Example datasets
x = [3, 5, 1, 8]
y = [1, 2, 3, 4]

# Calculate Euclidean Distance
distance = euclideanDistance(x, y)
print(f"Euclidean Distance: {distance}")
```

```
Euclidean Distance: 5.744562646538029
```

In [4]:
```
def manhattanDistance(x, y):
    return np.sum(np.abs(np.array(x) - np.array(y)))

# Example datasets
x = [3, 5, 1, 8]
y = [1, 2, 3, 4]

# Calculate Manhattan Distance
distance = manhattanDistance(x, y)
print(f"Manhattan Distance: {distance}")
```

```
Manhattan Distance: 11
```

In [5]:
```
def minkowskiDistance(x, y, p):
    return np.sum(np.abs(np.array(x) - np.array(y))**p)**(1/p)

# Example datasets
x = [3, 5, 1, 8]
y = [1, 2, 3, 4]
p = 3  # Minkowski distance parameter

# Calculate Minkowski Distance
distance = minkowskiDistance(x, y, p)
print(f"Minkowski Distance (p={p}): {distance}")
```

```
Minkowski Distance (p=3): 4.7474593985234
```

In [6]:
```
def hammingDistance(x, y):
    x, y = np.array(x), np.array(y)
    return np.sum(x != y)

# Example datasets (binary vectors)
x = [1, 0, 1, 1]
y = [0, 1, 1, 0]

# Calculate Hamming Distance
distance = hammingDistance(x, y)
print(f"Hamming Distance: {distance}")
```

```
Hamming Distance: 3
```

In [7]:
```
def cosineDistance(x, y):
    x = np.array(x)
    y = np.array(y)
    dot_product = np.dot(x, y)
    norm_x = np.linalg.norm(x)
    norm_y = np.linalg.norm(y)
    cosine_similarity = dot_product / (norm_x * norm_y)
    return 1 - cosine_similarity  # Cosine distance is 1 minus cosine simila

# Example datasets
x = [3, 5, 1, 8]
y = [1, 2, 3, 4]

# Calculate Cosine Distance
distance = cosineDistance(x, y)
print(f"Cosine Distance: {distance}")
```

```
Cosine Distance: 0.11922898789891156
```

In [8]:
```
def mahalanobisDistance(x, y, inv_cov_matrix):
    x = np.array(x)
    y = np.array(y)
    delta = x - y
    return np.sqrt(np.dot(np.dot(delta, inv_cov_matrix), delta.T))

# Example datasets
x = [3, 5, 1, 8]
y = [1, 2, 3, 4]

# Example dataset (used to compute the covariance matrix)
data = np.array([
    [3, 5, 1, 8],
    [1, 2, 3, 4],
    [5, 5, 5, 5],
    [10, 10, 10, 10]
])

# Compute covariance matrix
cov_matrix = np.cov(data, rowvar=False)

# Add regularization term (small value to the diagonal) to handle singular m
regularization_strength = 1e-10  # Small constant
cov_matrix += np.eye(cov_matrix.shape[0]) * regularization_strength

# Inverse of the regularized covariance matrix
inv_cov_matrix = np.linalg.inv(cov_matrix)

# Calculate Mahalanobis Distance
distance = mahalanobisDistance(x, y, inv_cov_matrix)
print(f"Mahalanobis Distance: {distance}")
```

```
Mahalanobis Distance: 2.4494910567915196
```

In [ ]: