In [138]:
```
#PRESENTADO POR:ANGIE PAOLA VILLADA ORTIZ 1089721336

#PROCESAMIENTO DIGITAL

# Se importan las librería numpy y las funciones de preprocesamiento
import numpy as np
from sklearn import preprocessing
# Datos de prueba
input_data = np.array([[7.0, -4.1, 5.1],
[-3.6, 9.5, -8.2],
[5.9, 2.2, 4.1],
[9.1, -10.7, -6.2]])

print(input_data)
```

```
[[  7.   -4.1   5.1]
 [ -3.6   9.5  -8.2]
 [  5.9   2.2   4.1]
 [  9.1 -10.7  -6.2]]
```

In [139]:
```
# Binarizar los datos

data_binarized = preprocessing.Binarizer(threshold=3.2).transform(input_data)
print("\nDatos binarizados:\n", data_binarized)
```

```
Datos binarizados:
 [[1. 0. 1.]
 [0. 1. 0.]
 [1. 0. 1.]
 [1. 0. 0.]]
```

In [140]:
```
# Imprimir la media y la desviación estándar

print("\nANTES:")
print("Media =", input_data.mean(axis=0))
print("Desviación estándar =", input_data.std(axis=0))
```

```
ANTES:
Media = [ 4.6   -0.775 -1.3  ]
Desviación estándar = [4.87185796 7.48310597 5.95273047]
```

In [141]:
```
# Remover la media

data_scaled = preprocessing.scale(input_data)
print("\nDESPUÉS:")
print("Media =", data_scaled.mean(axis=0))
print("Desviación estándar =", data_scaled.std(axis=0))
```

```
DESPUÉS:
Media = [ 8.32667268e-17  0.00000000e+00 -2.77555756e-17]
Desviación estándar = [1. 1. 1.]
```

In [142]:
```
# Escalamiento Min Max

data_scaler_minmax = preprocessing.MinMaxScaler(feature_range=(0,
1))
data_scaled_minmax = data_scaler_minmax.fit_transform(input_data)
print("\nMin max escalamiento de datos:\n", data_scaled_minmax)
```

```
Min max escalamiento de datos:
 [[0.83464567 0.32673267 1.        ]
 [0.         1.         0.        ]
 [0.7480315  0.63861386 0.92481203]
 [1.         0.         0.15037594]]
```

In [143]:
```
# Normalización de datos

data_normalized_l1 = preprocessing.normalize(input_data,norm='l1')
data_normalized_l2 = preprocessing.normalize(input_data,norm='l2')
print("\nL1 dato normalizado:\n", data_normalized_l1)
print("\nL2 dato normalizado:\n", data_normalized_l2)
```

```
L1 dato normalizado:
 [[ 0.43209877 -0.25308642  0.31481481]
 [-0.16901408  0.44600939 -0.38497653]
 [ 0.48360656  0.18032787  0.33606557]
 [ 0.35       -0.41153846 -0.23846154]]

L2 dato normalizado:
 [[ 0.73051543 -0.42787333  0.53223267]
 [-0.27574268  0.7276543  -0.62808056]
 [ 0.78520232  0.29278731  0.54564907]
 [ 0.59268611 -0.69689466 -0.40380812]]
```

In [144]:

```
# Manejo de etiquetas
import numpy as np
from sklearn import preprocessing
# Se definen algunas etiquetas simples
input_labels = ['red', 'black', 'red', 'green', 'black', 'yellow',
'white']
# Se crea un codificador de etiquetas y se ajustan las etiquetas
encoder = preprocessing.LabelEncoder()
encoder.fit(input_labels)

# Se imprime el mapeo entre palabras y números
print("\nMapeo de etiquetas:")
for i, item in enumerate(encoder.classes_):
    print(item, '-->', i)

# Codificar un conjunto de etiquetas con el codificador
test_labels = ['black', 'green', 'white']
encoded_values = encoder.transform(test_labels)
print("\nLabels =", test_labels)
print("Encoded values =", list(encoded_values))

# Decodificar un conjunto de valores usando el codificador
encoded_values = [2, 0, 2, 4]
decoded_list = encoder.inverse_transform(encoded_values)
print("\nEncoded values =", encoded_values)
print("Decoded labels =", list(decoded_list))
```

```
Mapeo de etiquetas:
black --> 0
green --> 1
red --> 2
white --> 3
yellow --> 4

Labels = ['black', 'green', 'white']
Encoded values = [0, 1, 3]

Encoded values = [2, 0, 2, 4]
Decoded labels = ['red', 'black', 'red', 'yellow']
```