In [1]:

```python
import pandas as pd

# Sample data
data = pd.DataFrame({
    'Gender': ['Male', 'Female', 'Female', 'Male', 'Male', 'Female'],
    'Department': ['HR', 'IT', 'Sales', 'HR', 'IT', 'Sales'],
    'Education': ['Bachelor', 'Master', 'PhD', 'Master', 'PhD', 'Bachelor'],
    'Salary': [50000, 60000, 55000, 52000, 58000, 53000],
    'Experience': [1.5, 3.0, 4.2, 2.3, 3.7, 2.9]
})

data.head()

```

```
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\num
py\_distributor_init.py:30: UserWarning: loaded more than 1 DLL from .libs:
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\num
py\.libs\libopenblas.EL2C6PLE4ZYW3ECEVIV3OXXGRN2NRFM2.gfortran-win_amd64.dll
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\num
py\.libs\libopenblas.XWYDX2IKJW2NMTWSFYNGFUWKQU3LYTCZ.gfortran-win_amd64.dll
  warnings.warn("loaded more than 1 DLL from .libs:"
```

Out[1]:

	Gender	Department	Education	Salary	Experience
0	Male	HR	Bachelor	50000	1.5
1	Female	IT	Master	60000	3.0
2	Female	Sales	PhD	55000	4.2
3	Male	HR	Master	52000	2.3
4	Male	IT	PhD	58000	3.7

In [2]:

```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Gender      6 non-null      object
 1   Department  6 non-null      object
 2   Education   6 non-null      object
 3   Salary      6 non-null      int64
 4   Experience  6 non-null      float64
dtypes: float64(1), int64(1), object(3)
memory usage: 368.0+ bytes
```

In [12]:

```
#Use LabelEncoder to convert categories into integers
from sklearn.preprocessing import LabelEncoder

# Make a copy to preserve original
encoded_data = data.copy()

le = LabelEncoder()
encoded_data['Gender'] = le.fit_transform(data['Gender'])
encoded_data['Department'] = le.fit_transform(data['Department'])
encoded_data['Education'] = le.fit_transform(data['Education'])
encoded_data.head()
encoded_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Gender      6 non-null      int32  
 1   Department  6 non-null      int32  
 2   Education   6 non-null      int32  
 3   Salary      6 non-null      int64  
 4   Experience  6 non-null      float64
dtypes: float64(1), int32(3), int64(1)
memory usage: 296.0 bytes
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype   
---  ------      --------------  -----   
 0   Gender      6 non-null      category
 1   Department  6 non-null      int32   
 2   Education   6 non-null      int32   
 3   Salary      6 non-null      int64   
 4   Experience  6 non-null      float64 
dtypes: category(1), float64(1), int32(2), int64(1)
memory usage: 402.0 bytes
```

In [13]:

```
encoded_data['Gender']=encoded_data['Gender'].astype('category')
encoded_data['Department']=encoded_data['Department'].astype('category')
encoded_data['Education']=encoded_data['Education'].astype('category')
encoded_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype   
---  ------      --------------  -----   
 0   Gender      6 non-null      category
 1   Department  6 non-null      category
 2   Education   6 non-null      category
 3   Salary      6 non-null      int64   
 4   Experience  6 non-null      float64 
dtypes: category(3), float64(1), int64(1)
memory usage: 630.0 bytes
```

In [20]:
```python
#One-Hot Encoding Use pd.get_dummies() to create binary columns.

onehot_encoded_data = pd.get_dummies(data, columns=['Gender', 'Department',

onehot_encoded_data

```

Out[20]:

	Salary	Experience	Gender_Female	Gender_Male	Department_HR	Department_IT	Department_Sa
0	50000	1.5	0	1	1	0	
1	60000	3.0	1	0	0	1	
2	55000	4.2	1	0	0	0	
3	52000	2.3	0	1	1	0	
4	58000	3.7	0	1	0	1	
5	53000	2.9	1	0	0	0	

In [8]:
```python
from sklearn.preprocessing import LabelBinarizer

lb = LabelBinarizer()
en_data=data.copy()
# Gender (Binary)
en_data['Gender'] = lb.fit_transform(data['Gender'])
en_data['Department'] = lb.fit_transform(data['Department'])
en_data['Education'] = lb.fit_transform(data['Education'])

en_data

```

Out[8]:

	Gender	Department	Education	Salary	Experience
0	1	1	1	50000	1.5
1	0	0	0	60000	3.0
2	0	0	0	55000	4.2
3	1	1	0	52000	2.3
4	1	0	0	58000	3.7
5	0	0	1	53000	2.9

In [9]:

```
# Department (Multiclass)
dept_binarized = pd.DataFrame(LabelBinarizer().fit_transform(data['Departmen
                              columns=['Dept_HR', 'Dept_IT', 'Dept_Sales'])
dept_binarized
```

Out[9]:

	Dept_HR	Dept_IT	Dept_Sales
0	1	0	0
1	0	1	0
2	0	0	1
3	1	0	0
4	0	1	0
5	0	0	1

In [10]:

```
# Education (Multiclass)
edu_binarized = pd.DataFrame(LabelBinarizer().fit_transform(data['Education'
                             columns=['Edu_Bachelor', 'Edu_Master', 'Edu_PhD

edu_binarized
```

Out[10]:

	Edu_Bachelor	Edu_Master	Edu_PhD
0	1	0	0
1	0	1	0
2	0	0	1
3	0	1	0
4	0	0	1
5	1	0	0

Similarity Measures

In [17]:

```
import numpy as np

# Two sample vectors
A = np.array([3.7, 2.90,9.87])
B = np.array([5.0, 6.5,8.6])
c=np.array([3.7,0,0])

```

1. Euclidean Distance (L2 norm)

Straight-line distance between points.

used: Good for continuous features, geometric closeness.

In [19]:
```
from scipy.spatial.distance import euclidean

print("Euclidean Distance:", euclidean(A, B,c))

```

```
Euclidean Distance: 2.5005999280172744
```

2. Manhattan Distance (L1 norm)

Sum of absolute differences.

Used in urban planning or anytime "right-angle" travel is needed.

In [21]:
```
from scipy.spatial.distance import cityblock

print("Manhattan Distance:", cityblock(A, B))

```

```
Manhattan Distance: 6.17
```

3. Cosine Similarity

Measures angle between vectors (not magnitude).

Ideal for text similarity, TF-IDF, embeddings.

In [23]:
```
from sklearn.metrics.pairwise import cosine_similarity

cos_sim = cosine_similarity([A], [B])
print("Cosine Similarity:", cos_sim[0][0])

```

```
Cosine Similarity: 0.9408871372284504
```

4. Jaccard Similarity

Works with sets or binary data.

Great for recommendations (e.g., common likes), binary vectors.

In [24]:
```
A_set = set([1, 2, 3])
B_set = set([2, 3, 4])

jaccard = len(A_set & B_set) / len(A_set | B_set)
print("Jaccard Similarity:", jaccard)

```

```
Jaccard Similarity: 0.5
```

5.Hamming Distance

Number of positions with different values (binary or strings).

Best for comparing binary vectors or DNA sequences.

In [34]:
```python
from scipy.spatial.distance import hamming

A_bin = np.array([1, 0, 1, 1])
B_bin = np.array([1, 1, 0, 1])

print("Hamming Distance:", hamming(A_bin, B_bin))

```

```
Hamming Distance: 0.5
```

****

1. Cosine Similarity on Text

Cosine Similarity works best when we vectorize the text (e.g., using TF-IDF or CountVectorizer).

Using TfidfVectorizer

In [28]:
```python
text1 = "I love machine learning"
text2 = "I enjoy learning machines"

```

In [29]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Sample texts
texts = [text1, text2]

# Vectorize
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(texts)

# Show feature names (unique words)
print(vectorizer.get_feature_names_out())

# Show the TF-IDF matrix as an array
tf=pd.DataFrame(tfidf_matrix.toarray(),columns=vectorizer.get_feature_names_
tf
```

```
['enjoy' 'learning' 'love' 'machine' 'machines']
```

Out[29]:

	enjoy	learning	love	machine	machines
0	0.000000	0.449436	0.631667	0.631667	0.000000
1	0.631667	0.449436	0.000000	0.000000	0.631667

In [30]:
```
# Compute Cosine Similarity
cos_sim = cosine_similarity(tfidf_matrix[0:1], tfidf_matrix[1:2])

print("Cosine Similarity:", cos_sim[0][0])

```

```
Cosine Similarity: 0.20199309249791833
```

In [ ]:
```

```

In [ ]:
```

```

🔁 2. Hamming Distance on Text

Hamming Distance only works on equal-length binary strings or character sequences.

So, we'll:

Convert each sentence to a fixed-length binary or character sequence

Pad if needed

Convert to character sequence (basic example)

In [31]:

```
from scipy.spatial.distance import hamming

# Shortened equal-length strings
text1 = "machine"
text2 = "mashing"

# Ensure equal length
min_len = min(len(text1), len(text2))
text1 = text1[:min_len]
text2 = text2[:min_len]

# Convert to list of characters
list1 = list(text1)
list2 = list(text2)

# Compute Hamming Distance
hamm_dist = hamming(list1, list2)
print("Hamming Distance:", hamm_dist)

```

```
Hamming Distance: 0.2857142857142857
```

In [ ]:

```
# # i.e., 2 out of 7 characters are different
# Value is normalized (0 = same, 1 = completely different)
```