In [3]:
```
import nltk
```

In [4]:
```
nltk.download('punkt')  # For tokenization
nltk.download('stopwords')  # For stopwords
nltk.download('wordnet')  # For lemmatization
nltk.download('averaged_perceptron_tagger')  # For POS tagging
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Welcome\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Welcome\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Welcome\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\Welcome\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

Out[4]: True

In [5]:
```
 #Step 2: Initialize Text
 text = "Natural Language Processing helps computers understand human language.
```

In [7]:
```
from nltk.tokenize import sent_tokenize

text = "Natural Language Processing helps computers understand human language."

# Sentence Tokenization
tokenized_text = sent_tokenize(text)
print(tokenized_text)
```

```
['Natural Language Processing helps computers understand human language.']
```

In [9]:
```
from nltk.tokenize import word_tokenize

text = "Natural Language Processing helps computers understand human language."

# Word Tokenization
tokenized_word = word_tokenize(text)
print(tokenized_word)
```

```
['Natural', 'Language', 'Processing', 'helps', 'computers', 'understand', 'hum
an', 'language', '.']
```

In [10]:

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import re

# Define stop words
stop_words = set(stopwords.words("english"))

# Sample text
text = "How to remove stop words with NLTK library in Python?"

# Remove punctuation (keep only letters)
text = re.sub('[^a-zA-Z]', ' ', text)

# Tokenize text
tokens = word_tokenize(text.lower())

# Remove stopwords
filtered_text = [w for w in tokens if w not in stop_words]

# Output results
print("Tokenized Sentence:", tokens)
print("Filtered Sentence:", filtered_text)
```

```
Tokenized Sentence: ['how', 'to', 'remove', 'stop', 'words', 'with', 'nltk',
'library', 'in', 'python']
Filtered Sentence: ['remove', 'stop', 'words', 'nltk', 'library', 'python']
```

In [11]:

```
from nltk.stem import PorterStemmer

# List of words to stem
e_words = ["wait", "waiting", "waited", "waits"]

# Initialize the stemmer
ps = PorterStemmer()

# Perform stemming
for w in e_words:
    rootWord = ps.stem(w)
    print(rootWord)
```

```
wait
wait
wait
wait
```

In [12]:
```
from nltk.stem import WordNetLemmatizer
import nltk

# Initialize the Lemmatizer
wordnet_lemmatizer = WordNetLemmatizer()

# Sample text
text = "studies studying cries cry"

# Tokenize the text
tokenization = nltk.word_tokenize(text)

# Perform Lemmatization
for w in tokenization:
    print("Lemma for {}: {}".format(w, wordnet_lemmatizer.lemmatize(w)))
```

```
Lemma for studies: study
Lemma for studying: studying
Lemma for cries: cry
Lemma for cry: cry
```

In [13]:
```
from nltk.tokenize import word_tokenize
from nltk import pos_tag

# Sample text
data = "The pink sweater fit her perfectly"

# Tokenize the text
words = word_tokenize(data)

# Apply POS tagging
pos_tags = pos_tag(words)

# Print the results
print(pos_tags)
```

```
[('The', 'DT'), ('pink', 'NN'), ('sweater', 'NN'), ('fit', 'VBP'), ('her', 'PR
P$'), ('perfectly', 'RB')]
```

In [14]:
```
#Part 2: TF-IDF Representation of Document
```

In [16]:
```
# Step 1: Import Required Libraries
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
import math
```

In [17]:
```
# Step 2: Initialize the Documents
documentA = "Jupiter is the largest Planet"
documentB = "Mars is the fourth planet from the Sun"
```

In [19]:
```
# Step 3: Create Bag of Words (BoW) for Document A and B
bagOfWordsA = documentA.split(' ')
bagOfWordsB = documentB.split(' ')
```

In [20]:
```
bagOfWordsA
```

Out[20]:
```
['Jupiter', 'is', 'the', 'largest', 'Planet']
```

In [21]:
```
bagOfWordsB
```

Out[21]:
```
['Mars', 'is', 'the', 'fourth', 'planet', 'from', 'the', 'Sun']
```

In [22]:
```
# Step 4: Create Collection of Unique Words from Document A and B
uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))
print(uniqueWords)
```

```
{'from', 'is', 'Sun', 'largest', 'the', 'fourth', 'Mars', 'planet', 'Planet',
'Jupiter'}
```

In [23]:
```
# Step 5: Create a Dictionary of Words and Their Occurrence for Each Document
numOfWordsA = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsA:
    numOfWordsA[word] += 1

numOfWordsB = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsB:
    numOfWordsB[word] += 1

# Print the word occurrence dictionaries
print("Word Occurrences in Document A:", numOfWordsA)
print("Word Occurrences in Document B:", numOfWordsB)
```

```
Word Occurrences in Document A: {'from': 0, 'is': 1, 'Sun': 0, 'largest': 1,
'the': 1, 'fourth': 0, 'Mars': 0, 'planet': 0, 'Planet': 1, 'Jupiter': 1}
Word Occurrences in Document B: {'from': 1, 'is': 1, 'Sun': 1, 'largest': 0,
'the': 2, 'fourth': 1, 'Mars': 1, 'planet': 1, 'Planet': 0, 'Jupiter': 0}
```

In [24]:
```
numOfWordsA
```

Out[24]:
```
{'from': 0,
 'is': 1,
 'Sun': 0,
 'largest': 1,
 'the': 1,
 'fourth': 0,
 'Mars': 0,
 'planet': 0,
 'Planet': 1,
 'Jupiter': 1}
```

In [25]:
```
numOfWordsB
```

Out[25]:
```
{'from': 1,
 'is': 1,
 'Sun': 1,
 'largest': 0,
 'the': 2,
 'fourth': 1,
 'Mars': 1,
 'planet': 1,
 'Planet': 0,
 'Jupiter': 0}
```

In [26]:
```
# Step 6: Compute Term Frequency (TF)
def computeTF(wordDict, bagOfWords):
    tfDict = {}
    bagOfWordsCount = len(bagOfWords)
    for word, count in wordDict.items():
        tfDict[word] = count / float(bagOfWordsCount)
    return tfDict

# Compute TF for both Document A and B
tfA = computeTF(numOfWordsA, bagOfWordsA)
tfB = computeTF(numOfWordsB, bagOfWordsB)

# Print Term Frequency (TF) for each document
print("Term Frequency for Document A:", tfA)
print("Term Frequency for Document B:", tfB)
```

```
Term Frequency for Document A: {'from': 0.0, 'is': 0.2, 'Sun': 0.0, 'largest': 0.2, 'the': 0.2, 'fourth': 0.0, 'Mars': 0.0, 'planet': 0.0, 'Planet': 0.2, 'Jupiter': 0.2}
Term Frequency for Document B: {'from': 0.125, 'is': 0.125, 'Sun': 0.125, 'largest': 0.0, 'the': 0.25, 'fourth': 0.125, 'Mars': 0.125, 'planet': 0.125, 'Planet': 0.0, 'Jupiter': 0.0}
```

In [27]:
```
tfA
```

Out[27]:
```
{'from': 0.0,
 'is': 0.2,
 'Sun': 0.0,
 'largest': 0.2,
 'the': 0.2,
 'fourth': 0.0,
 'Mars': 0.0,
 'planet': 0.0,
 'Planet': 0.2,
 'Jupiter': 0.2}
```

In [28]:
```
tfB
```

Out[28]:
```
{'from': 0.125,
 'is': 0.125,
 'Sun': 0.125,
 'largest': 0.0,
 'the': 0.25,
 'fourth': 0.125,
 'Mars': 0.125,
 'planet': 0.125,
 'Planet': 0.0,
 'Jupiter': 0.0}
```

In [29]:
```
# Step 7: Compute Inverse Document Frequency (IDF)
def computeIDF(documents):
    N = len(documents)
    idfDict = dict.fromkeys(documents[0].keys(), 0)
    for document in documents:
        for word, val in document.items():
            if val > 0:
                idfDict[word] += 1
    for word, val in idfDict.items():
        idfDict[word] = math.log(N / float(val))
    return idfDict

# Compute IDF for the documents
idfs = computeIDF([numOfWordsA, numOfWordsB])

# Print Inverse Document Frequency (IDF)
print("Inverse Document Frequency (IDF):", idfs)
```

```
Inverse Document Frequency (IDF): {'from': 0.6931471805599453, 'is': 0.0, 'Su
n': 0.6931471805599453, 'largest': 0.6931471805599453, 'the': 0.0, 'fourth':
0.6931471805599453, 'Mars': 0.6931471805599453, 'planet': 0.6931471805599453,
'Planet': 0.6931471805599453, 'Jupiter': 0.6931471805599453}
```

In [30]:
```
idfs
```

Out[30]:
```
{'from': 0.6931471805599453,
 'is': 0.0,
 'Sun': 0.6931471805599453,
 'largest': 0.6931471805599453,
 'the': 0.0,
 'fourth': 0.6931471805599453,
 'Mars': 0.6931471805599453,
 'planet': 0.6931471805599453,
 'Planet': 0.6931471805599453,
 'Jupiter': 0.6931471805599453}
```

In [31]:

```
# Step 8: Compute TF-IDF
def computeTFIDF(tfBagOfWords, idfs):
    tfidf = {}
    for word, val in tfBagOfWords.items():
        tfidf[word] = val * idfs[word]
    return tfidf

# Compute TF-IDF for both Document A and B
tfidfA = computeTFIDF(tfA, idfs)
tfidfB = computeTFIDF(tfB, idfs)

# Create a DataFrame for visualization
df = pd.DataFrame([tfidfA, tfidfB])

# Print the DataFrame
print(df)
```

```
       from   is       Sun   largest  the    fourth      Mars    planet  \
0  0.000000  0.0  0.000000  0.138629  0.0  0.000000  0.000000  0.000000
1  0.086643  0.0  0.086643  0.000000  0.0  0.086643  0.086643  0.086643

     Planet   Jupiter
0  0.138629  0.138629
1  0.000000  0.000000
```

In [ ]:

```
                                    NAME:SHARVARI PATIL
                                    ROLL NO:13265
                                    BATCH:B3
```