In [18]:
```
import nltk
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
nltk.download('stopwords')
from nltk import sent_tokenize
from nltk import word_tokenize
from nltk.corpus import stopwords
```

```
[nltk_data] Downloading package punkt to /Users/anvi/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /Users/anvi/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /Users/anvi/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package stopwords to /Users/anvi/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [19]:
```
print('Text:')
print()
text1="I will walk 500 miles and would walk 500 more.    Just to walk a 1000
text2="I played the act of play nicely and playfully as the players were pla
print("Text1:  "+text1)
print()
print("Text2:  "+text2)
```

```
Text:

Text1:  I will walk 500 miles and would walk 500 more.    Just to walk a 100
0 miles till your door !

Text2:  I played the act of play nicely and playfully as the players were pl
aying their parts in the play
```

In [20]:
```
print('Tokenization:')
print()
print()
print('Tokenized words:',word_tokenize(text1))
print()
print('Tokenized sentences',sent_tokenize(text1))
print()
print('Tokenized words:',word_tokenize(text2))
print()
print('Tokenized sentences',sent_tokenize(text2))
```

```
Tokenization:


Tokenized words: ['I', 'will', 'walk', '500', 'miles', 'and', 'would', 'wal
k', '500', 'more', '.', 'Just', 'to', 'walk', 'a', '1000', 'miles', 'till',
'your', 'door', '!']

Tokenized sentences ['I will walk 500 miles and would walk 500 more.', 'Just
to walk a 1000 miles till your door !']

Tokenized words: ['I', 'played', 'the', 'act', 'of', 'play', 'nicely', 'an
d', 'playfully', 'as', 'the', 'players', 'were', 'playing', 'their', 'part
s', 'in', 'the', 'play']

Tokenized sentences ['I played the act of play nicely and playfully as the p
layers were playing their parts in the play']
```

In [21]:
```
print('POS Tagging:')
print()
print()
from nltk import pos_tag
token=word_tokenize(text1)+word_tokenize(text2)
tagged=pos_tag(token)

print('Tagging parts of speech:',tagged)
print()
```

```
POS Tagging:


Tagging parts of speech: [('I', 'PRP'), ('will', 'MD'), ('walk', 'VB'), ('50
0', 'CD'), ('miles', 'NNS'), ('and', 'CC'), ('would', 'MD'), ('walk', 'VB'),
('500', 'CD'), ('more', 'JJR'), ('.', '.'), ('Just', 'RB'), ('to', 'TO'),
('walk', 'VB'), ('a', 'DT'), ('1000', 'CD'), ('miles', 'NNS'), ('till', 'R
B'), ('your', 'PRP$'), ('door', 'NN'), ('!', '.'), ('I', 'PRP'), ('played',
'VBD'), ('the', 'DT'), ('act', 'NN'), ('of', 'IN'), ('play', 'NN'), ('nicel
y', 'RB'), ('and', 'CC'), ('playfully', 'RB'), ('as', 'IN'), ('the', 'DT'),
('players', 'NNS'), ('were', 'VBD'), ('playing', 'VBG'), ('their', 'PRP$'),
('parts', 'NNS'), ('in', 'IN'), ('the', 'DT'), ('play', 'NN')]
```

In [22]:
```
print('Stop Words Removal') #stopwords are the,is,and
from nltk.corpus import stopwords

stop_words=stopwords.words('english')
token=word_tokenize(text1)
cleaned_token=[]

for word in token:
    if word not in stop_words:
        cleaned_token.append(word)
print('Uncleaned version:', token)
print()
print('Cleaned version', cleaned_token)
```

```
Stop Words Removal
Uncleaned version: ['I', 'will', 'walk', '500', 'miles', 'and', 'would', 'wa
lk', '500', 'more', '.', 'Just', 'to', 'walk', 'a', '1000', 'miles', 'till',
'your', 'door', '!']

Cleaned version ['I', 'walk', '500', 'miles', 'would', 'walk', '500', '.',
'Just', 'walk', '1000', 'miles', 'till', 'door', '!']
```

In [23]:
```python
#Stemming
from nltk.stem import PorterStemmer
stemmer=PorterStemmer()
token=word_tokenize(text2)
stemmed=[stemmer.stem(word)for word in token]
print(" ".join(stemmed))
```

```
i play the act of play nice and play as the player were play their part in t
he play
```

In [24]:
```python
#Lemmatization
from nltk.stem import WordNetLemmatizer
lemmatizer=WordNetLemmatizer ()
token=word_tokenize(text2)
lemmatized_output=[lemmatizer.lemmatize(word) for word in token]
print(" ".join(lemmatized_output))
```

```
I played the act of play nicely and playfully a the player were playing thei
r part in the play
```

In [25]:
```python
corpus=['the play was appreciated by players, act, and audience. the act was
```

In [26]:
```python
words_set=set()

for doc in corpus:
    words=doc.split(' ')
    words_set=words_set.union(set(words))
print('No of words in corpus:', len(words_set))
print('Words in the corpus are:\n',words_set)
```

```
No of words in corpus: 11
Words in the corpus are:
 {'by', 'players,', 'act', 'audience.', 'play', 'and', 'good', 'was', 'appre
ciated', 'act,', 'the'}
```

In [27]:
```python
import pandas as pd
import numpy as np
n_docs=len(corpus)
n_words_set=len(words_set)
words_list=list(words_set)
df_tf=pd.DataFrame(np.zeros((n_docs, n_words_set)), columns=words_list)

for i in range(n_docs):
    words=corpus[i].split(' ')
    for w in words:
        df_tf[w][i]=df_tf[w][i]+(1/len(words))

df_tf
```

Out[27]:

	by	players,	act	audience.	play	and	good	was
0	0.076923	0.076923	0.076923	0.076923	0.076923	0.076923	0.076923	0.153846

In [28]:

```
import pandas as pd

# Example corpus
corpus = ["This is the first document.",
          "This document is the second document.",
          "And this is the third one.",
          "Is this the first document?"]

# Step 1: Calculate TF
# Create a DataFrame with term frequencies
df_tf = pd.DataFrame()

for doc in corpus:
    words = doc.lower().split()  # Split document into words
    word_counts = pd.Series(words).value_counts(normalize=True)  # Calculate
    df_tf = df_tf.append(word_counts, ignore_index=True)  # Append to DataFr

df_tf.fillna(0, inplace=True)  # Fill NaN values with 0
print("Term Frequency (TF):")
print(df_tf)


# Step 2: Calculate IDF
# Calculate document frequencies
df_idf = pd.Series()
for doc in corpus:
    words = set(doc.lower().split())  # Unique words in each document
    df_idf = df_idf.add(pd.Series(list(words)), fill_value=0)  # Add 1 for e

idf = np.log(len(corpus) / (1 + df_idf))  # Calculate IDF
print("\nInverse Document Frequency (IDF):")
print(idf)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/var/folders/5r/dmv946n139q8vvmj1hc60c000000gn/T/ipykernel_1378/1741231615.p
y in ?()
     12 
     13 for doc in corpus:
     14     words = doc.lower().split()  # Split document into words
     15     word_counts = pd.Series(words).value_counts(normalize=True)  # C
alculate term frequencies
---> 16     df_tf = df_tf.append(word_counts, ignore_index=True)  # Append t
o DataFrame
     17 
     18 df_tf.fillna(0, inplace=True)  # Fill NaN values with 0
     19 print("Term Frequency (TF):")

~/anaconda3/lib/python3.11/site-packages/pandas/core/generic.py in ?(self, n
ame)
   6200             and name not in self._accessors
   6201             and self._info_axis._can_hold_identifiers_and_holds_name
(name)
   6202         ):
   6203             return self[name]
-> 6204         return object.__getattribute__(self, name)

AttributeError: 'DataFrame' object has no attribute 'append'
```

In [29]:
```python
import pandas as pd

# Example corpus
corpus = ["This is the first document.",
          "This document is the second document.",
          "And this is the third one.",
          "Is this the first document?"]

# Step 1: Calculate TF
# Create a DataFrame with term frequencies
df_tf_list = []

for doc in corpus:
    words = doc.lower().split()  # Split document into words
    word_counts = pd.Series(words).value_counts(normalize=True)  # Calculate
    df_tf_list.append(word_counts)

df_tf = pd.concat(df_tf_list, axis=1).fillna(0).T  # Concatenate and transpo
print("Term Frequency (TF):")
print(df_tf)
```

```
Term Frequency (TF):
                this        is       the  first  document.  document  \
proportion  0.200000  0.200000  0.200000    0.2   0.200000  0.000000
proportion  0.166667  0.166667  0.166667    0.0   0.166667  0.166667
proportion  0.166667  0.166667  0.166667    0.0   0.000000  0.000000
proportion  0.200000  0.200000  0.200000    0.2   0.000000  0.000000

              second       and     third      one.  document?
proportion  0.000000  0.000000  0.000000  0.000000        0.0
proportion  0.166667  0.000000  0.000000  0.000000        0.0
proportion  0.000000  0.166667  0.166667  0.166667        0.0
proportion  0.000000  0.000000  0.000000  0.000000        0.2
```

In [30]:

```python
import pandas as pd
import numpy as np

# Example corpus
corpus = ["This is the first document.",
          "This document is the second document.",
          "And this is the third one.",
          "Is this the first document?"]

# Calculate document frequencies
doc_freq = {}
for doc in corpus:
    words = set(doc.lower().split())  # Unique words in each document
    for word in words:
        doc_freq[word] = doc_freq.get(word, 0) + 1  # Count document frequen

# Convert document frequencies to a Pandas Series
df_idf = pd.Series(doc_freq)

# Calculate IDF
idf = np.log(len(corpus) / (1 + df_idf))  # Calculate IDF

print("Inverse Document Frequency (IDF):")
print(idf)
```

```
Inverse Document Frequency (IDF):
first         0.287682
is           -0.223144
this         -0.223144
document.     0.287682
the          -0.223144
document      0.693147
second        0.693147
and           0.693147
one.          0.693147
third         0.693147
document?     0.693147
dtype: float64
```

In [ ]: