In [1]:
```
docA = "When Antony saw that Julius Caesar lay dead"
docB = "The world saw the demise of Julius Caesar"
docC = "Antony saw Julius Caesar lay dead"
docD = "It was him my cat"
```

In [2]:
```
from nltk.tokenize import TreebankWordTokenizer
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))

bowA = TreebankWordTokenizer().tokenize(docA)
bowB = TreebankWordTokenizer().tokenize(docB)
bowC = TreebankWordTokenizer().tokenize(docC)

nbowA = []
nbowB = []
nbowC = []

nbowD = docD.split(" ")

for i,j,k in zip(bowA,bowB,bowC):
    if i not in stop_words:
        nbowA.append(i)
    if j not in stop_words:
        nbowB.append(j)
    if k not in stop_words:
        nbowC.append(k)
```

In [3]:
```
wordSet = set(nbowA).union(set(nbowB)).union(set(nbowC)).union(set(nbow
```

In [4]:
```
wordSet
```

Out[4]:
```
{'Antony',
 'Caesar',
 'It',
 'Julius',
 'The',
 'When',
 'cat',
 'dead',
 'demise',
 'him',
 'lay',
 'my',
 'saw',
 'was',
 'world'}
```

In [5]:
```
wordDictA = dict.fromkeys(wordSet, 0)
wordDictB = dict.fromkeys(wordSet, 0)
wordDictC = dict.fromkeys(wordSet, 0)
wordDictD = dict.fromkeys(wordSet, 0)
```

In [6]:
```
for word in nbowA:
    wordDictA[word]+=1

for word in nbowB:
    wordDictB[word]+=1

for word in nbowC:
    wordDictC[word]+=1

for word in nbowD:
    wordDictD[word]+=1
```

In [7]:
```
print(wordDictA)
print(wordDictB)
print(wordDictC)
print(wordDictD)
```

```
{'was': 0, 'Julius': 1, 'Antony': 1, 'dead': 0, 'When': 1, 'world':
0, 'lay': 0, 'my': 0, 'saw': 1, 'demise': 0, 'The': 0, 'him': 0, 'It
': 0, 'cat': 0, 'Caesar': 1}
{'was': 0, 'Julius': 0, 'Antony': 0, 'dead': 0, 'When': 0, 'world':
1, 'lay': 0, 'my': 0, 'saw': 1, 'demise': 1, 'The': 1, 'him': 0, 'It
': 0, 'cat': 0, 'Caesar': 0}
{'was': 0, 'Julius': 1, 'Antony': 1, 'dead': 1, 'When': 0, 'world':
0, 'lay': 1, 'my': 0, 'saw': 1, 'demise': 0, 'The': 0, 'him': 0, 'It
': 0, 'cat': 0, 'Caesar': 1}
{'was': 1, 'Julius': 0, 'Antony': 0, 'dead': 0, 'When': 0, 'world':
0, 'lay': 0, 'my': 1, 'saw': 0, 'demise': 0, 'The': 0, 'him': 1, 'It
': 1, 'cat': 1, 'Caesar': 0}
```

In [8]:
```
import pandas as pd
pd.DataFrame([wordDictA, wordDictB, wordDictC, wordDictD])
```

Out[8]:

	Antony	Caesar	It	Julius	The	When	cat	dead	demise	him	lay	my	saw	was	wor
0	1	1	0	1	0	1	0	0	0	0	0	0	1	0	
1	0	0	0	0	1	0	0	0	1	0	0	0	1	0	
2	1	1	0	1	0	0	0	1	0	0	1	0	1	0	
3	0	0	1	0	0	0	1	0	0	1	0	1	0	1	

In [9]:

```
def computeTF(wordDict, bow):
    tfDict = {}
    bowCount = len(bow)
    for word, count in wordDict.items():
        tfDict[word] = count/float(bowCount)
    return tfDict
```

In [10]:

```
tfBowA = computeTF(wordDictA, nbowA)
tfBowB = computeTF(wordDictB, nbowB)
tfBowC = computeTF(wordDictC, nbowC)
tfBowD = computeTF(wordDictD, nbowD)
```

In [11]:

```
tfBowA
```

Out[11]:

```
{'was': 0.0,
 'Julius': 0.2,
 'Antony': 0.2,
 'dead': 0.0,
 'When': 0.2,
 'world': 0.0,
 'lay': 0.0,
 'my': 0.0,
 'saw': 0.2,
 'demise': 0.0,
 'The': 0.0,
 'him': 0.0,
 'It': 0.0,
 'cat': 0.0,
 'Caesar': 0.2}
```

In [12]:

```
tfBowB
```

Out[12]:

```
{'was': 0.0,
 'Julius': 0.0,
 'Antony': 0.0,
 'dead': 0.0,
 'When': 0.0,
 'world': 0.25,
 'lay': 0.0,
 'my': 0.0,
 'saw': 0.25,
 'demise': 0.25,
 'The': 0.25,
 'him': 0.0,
 'It': 0.0,
 'cat': 0.0,
 'Caesar': 0.0}
```

In [13]:

```
tfBowC
```

Out[13]:

```
{'was': 0.0,
 'Julius': 0.16666666666666666,
 'Antony': 0.16666666666666666,
 'dead': 0.16666666666666666,
 'When': 0.0,
 'world': 0.0,
 'lay': 0.16666666666666666,
 'my': 0.0,
 'saw': 0.16666666666666666,
 'demise': 0.0,
 'The': 0.0,
 'him': 0.0,
 'It': 0.0,
 'cat': 0.0,
 'Caesar': 0.16666666666666666}
```

In [14]:

```
tfBowD
```

Out[14]:

```
{'was': 0.2,
 'Julius': 0.0,
 'Antony': 0.0,
 'dead': 0.0,
 'When': 0.0,
 'world': 0.0,
 'lay': 0.0,
 'my': 0.2,
 'saw': 0.0,
 'demise': 0.0,
 'The': 0.0,
 'him': 0.2,
 'It': 0.2,
 'cat': 0.2,
 'Caesar': 0.0}
```

In [15]:

```
def computeIDF(docList):
    import math
    idfDict = {}
    N = len(docList)

    idfDict = dict.fromkeys(docList[0].keys(), 0)
    for doc in docList:
        for word, val in doc.items():
            if val > 0:
                idfDict[word] += 1

    for word, val in idfDict.items():
        idfDict[word] = math.log10(N / float(val))

    return idfDict
```

In [16]:
```
idfs = computeIDF([wordDictA, wordDictB, wordDictC, wordDictD])
```

In [17]:
```
def computeTFIDF(tfBow, idfs):
    tfidf = {}
    for word, val in tfBow.items():
        tfidf[word] = val*idfs[word]
    return tfidf
```

In [18]:
```
tfidfBowA = computeTFIDF(tfBowA, idfs)
tfidfBowB = computeTFIDF(tfBowB, idfs)
tfidfBowC = computeTFIDF(tfBowC, idfs)
tfidfBowD = computeTFIDF(tfBowD, idfs)
```

In [19]:
```
import pandas as pd
D = pd.DataFrame([tfidfBowA, tfidfBowB, tfidfBowC, tfidfBowD])
```

In [20]:
```
D
```

Out[20]:

	Antony	Caesar	It	Julius	The	When	cat	dead	demise
0	0.060206	0.060206	0.000000	0.060206	0.000000	0.120412	0.000000	0.000000	0.000000
1	0.000000	0.000000	0.000000	0.000000	0.150515	0.000000	0.000000	0.000000	0.150515
2	0.050172	0.050172	0.000000	0.050172	0.000000	0.000000	0.000000	0.100343	0.000000
3	0.000000	0.000000	0.120412	0.000000	0.000000	0.000000	0.120412	0.000000	0.000000

In [21]:
```
A = list(D.iloc[0])
B = list(D.iloc[1])
C = list(D.iloc[2])
D = list(D.iloc[3])
```

In [22]:
```
from scipy import spatial

result = 1 - spatial.distance.cosine(A,B)
```

In [23]:
```
#Cosine Similarity between Doc A and B
result
```

Out[23]:
```
0.018435768212138326
```

In [24]:
```
#Cosine Similarity between Doc A and C
result = 1 - spatial.distance.cosine(A,C)
result
```

Out[24]:
```
0.3543805745724091
```

```
In [25]: #Cosine Similarity between Doc A and D
         result = 1 - spatial.distance.cosine(A,D)
         result

Out[25]: 0.0

In [ ]:

In [ ]:
```