```
import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
from nltk.tokenize import word_tokenize
import docx
import pandas as pd
import math
```

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

```
word_Doc=docx.Document('sample_data.docx')
```

```
extractedText = word_Doc.paragraphs[0].text
```

```
tokens = word_tokenize("The quick brown fox jumps over the lazy dog")
nltk.download('stopwords')
print(tokens)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']
```

```
tag = nltk.pos_tag(tokens)
print(tag)
```

```
[('The', 'DT'), ('quick', 'JJ'), ('brown', 'NN'), ('fox', 'NN'), ('jumps', 'VBZ'), (
```

```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
tokens = [w for w in tokens if not w in stop_words]
print(tokens)
```

```
['The', 'quick', 'brown', 'fox', 'jumps', 'lazy', 'dog']
```

```
from nltk.stem.porter import PorterStemmer
porter = PorterStemmer()
stems = []
for t in tokens:
    stems.append(porter.stem(t))
print(stems)
```

```
['the', 'quick', 'brown', 'fox', 'jump', 'lazi', 'dog']
```

```
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

for w in tokens:
  print(lemmatizer.lemmatize(w))
```

```
The
quick
brown
fox
jump
lazy
dog
```

```
first_sentence = word_Doc.paragraphs[1].text
second_sentence = word_Doc.paragraphs[2].text
```

```
first_sentence = first_sentence.split(" ")
second_sentence = second_sentence.split(" ")#join them to remove common duplicate words
total= set(first_sentence).union(set(second_sentence))
print(total)
```

```
{'job', 'machine', 'key', 'best', 'Data', 'the', 'of', '21st', 'is', 'science', 'for
```

```
wordDictA = dict.fromkeys(total, 0)
wordDictB = dict.fromkeys(total, 0)
for word in first_sentence:
    wordDictA[word]+=1

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

print(pd.DataFrame([wordDictA, wordDictB]))
```

```
   job  machine  key  best  Data  the  of  21st  is  science  for  Science  \
0    1        0    0     1     1    2   1     1   1        0    0        1
1    0        1    1     0     0    1   0     0   1        1    1        0

   century  learning  data
0        1         0     0
1        0         1     1
```

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

#running our sentences through the tf function:
```

```
tfFirst = computeTF(wordDictA, first_sentence)
tfSecond = computeTF(wordDictB, second_sentence)

#Converting to dataframe for visualization
tf = pd.DataFrame([tfFirst, tfSecond])

print(tf)
```

```
   job  machine    key  best  Data    the   of  21st     is  science    for  \
0  0.1    0.000  0.000   0.1   0.1  0.200  0.1   0.1  0.100    0.000  0.000
1  0.0    0.125  0.125   0.0   0.0  0.125  0.0   0.0  0.125    0.125  0.125

   Science  century  learning   data
0      0.1      0.1     0.000  0.000
1      0.0      0.0     0.125  0.125
```

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

    idfDict = dict.fromkeys(docList[0].keys(), 0)
    for word, val in idfDict.items():
        idfDict[word] = math.log10(N / (float(val) + 1))

    return(idfDict)
#inputing our sentences in the log file
idfs = computeIDF([wordDictA, wordDictB])
print(idfs)
```

```
56639812, 'Data': 0.3010299956639812, 'the': 0.3010299956639812, 'of': 0.3010299956639
```