TEXT PROCESSING

OPTICAL CHARACTER RECOGNITION

ABSTRACT

Optical character recognition or optical character reader (OCR) is the electronic or mechanical conversion of images of typed, handwritten, or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo or from subtitle text superimposed on an image. In many different fields, there is a high demand for storing information to a computer storage disk from the data available in printed or handwritten documents or images to later re-utilize this information by means of computers. One simple way to store information to a computer system from these printed documents could be first to scan the documents and then store them as image files. But to re-utilize this information, it would very difficult to read or query text or other information from these image files. Therefore a technique to automatically retrieve and store information, in particular text, from image files is needed. Optical character recognition is an active research area that attempts to develop a computer system with the ability to extract and process text from images automatically. The objective of OCR is to achieve modification or conversion of any form of text or text-containing documents such as handwritten text, printed or scanned text images, into an editable digital format for deeper and further processing. Therefore, OCR enables a machine to automatically recognize text in such documents.

DATA COLLECTION

The dataset which is used in the project is IAM HANDWRITTEN FROMS DATASET which is open-source dataset which can be download from the Kaggle. The dataset contains complete forms of unconstrained handwritten text, which were scanned at a resolution of 300dpi and saved as PNG images with 256 gray levels. Forms are partitioned into separate directories such that all forms in each directory are written by the same person.

STEPS TO FOLLOW BEFORE DATA PREPROCESSING AND FEATURE EXTRACTION

1.Signup for Kaggle to get access to dataset

2.Follow the below link to download the dataset:

https://www.kaggle.com/datasets/naderabdalghani/iam-handwritten-forms-dataset

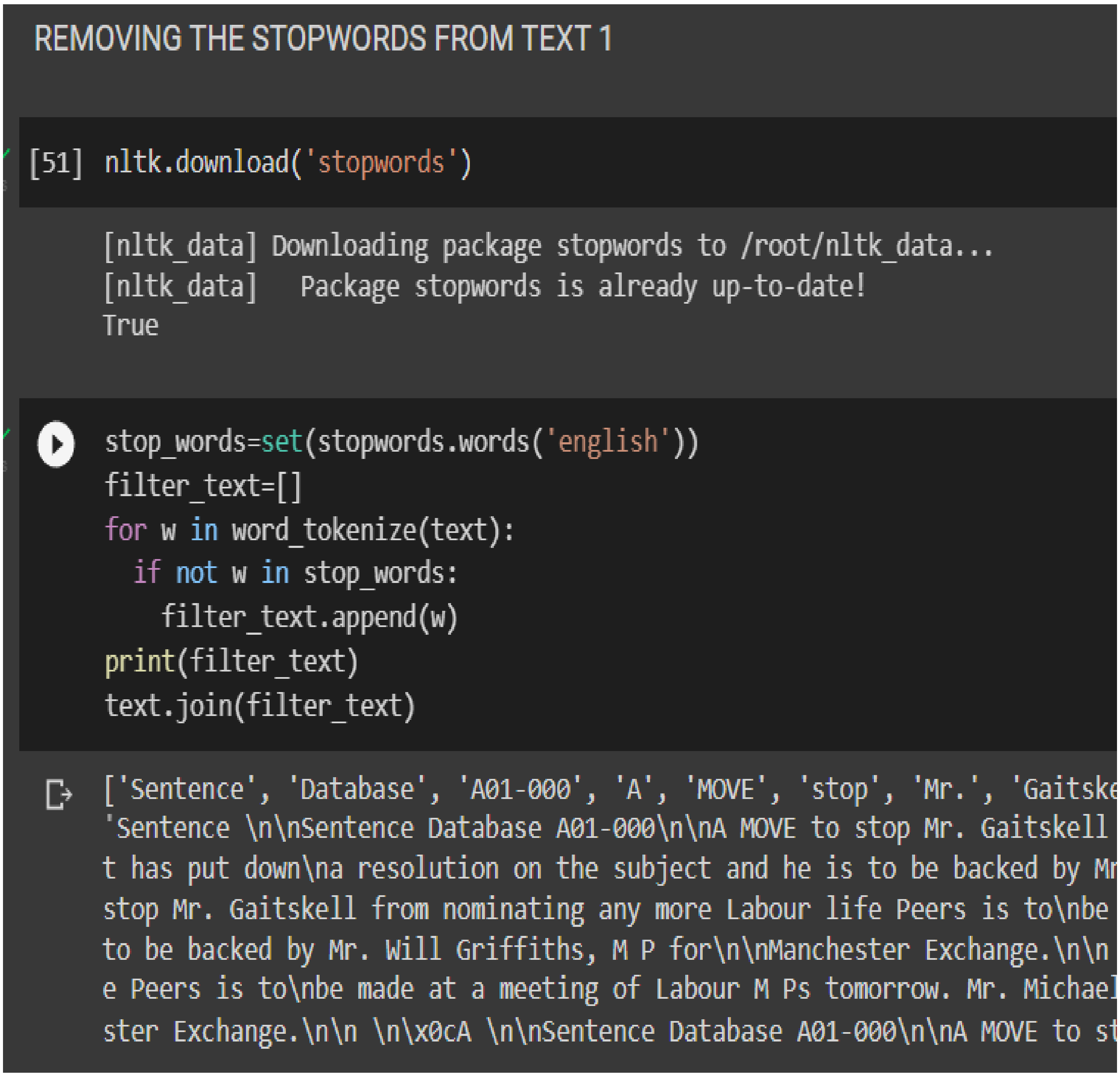
3.After downloading the dataset extract it and upload it to drive.

4.Mount the drive and import image for further process.

5.Convert the image to text using pytesseract.

DATA PREPROCESSING AND FEATURE EXTRACTION

After converting the image to text, we delete stop words and cleaned our text for tokenization and stemming.



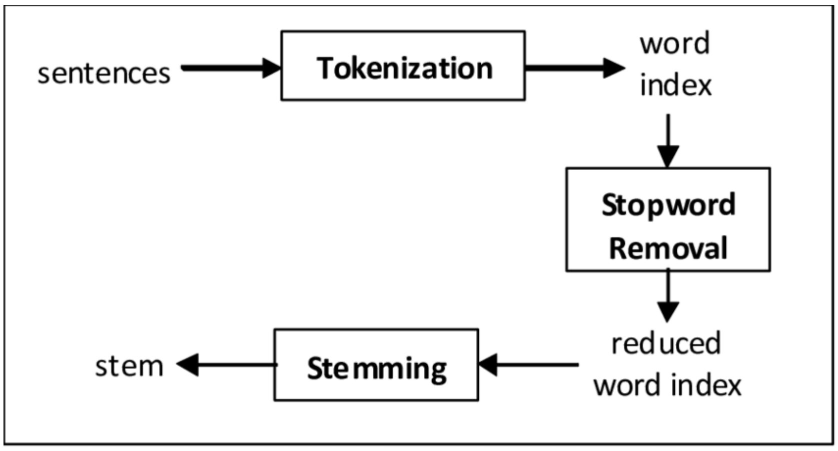
REMOVING THE STOPWORDS FROM TEXT 1

```
[51] nltk.download('stopwords')

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

```
stop_words=set(stopwords.words('english'))
filter_text=[]
for w in word_tokenize(text):
  if not w in stop_words:
    filter_text.append(w)
print(filter_text)
text.join(filter_text)
```

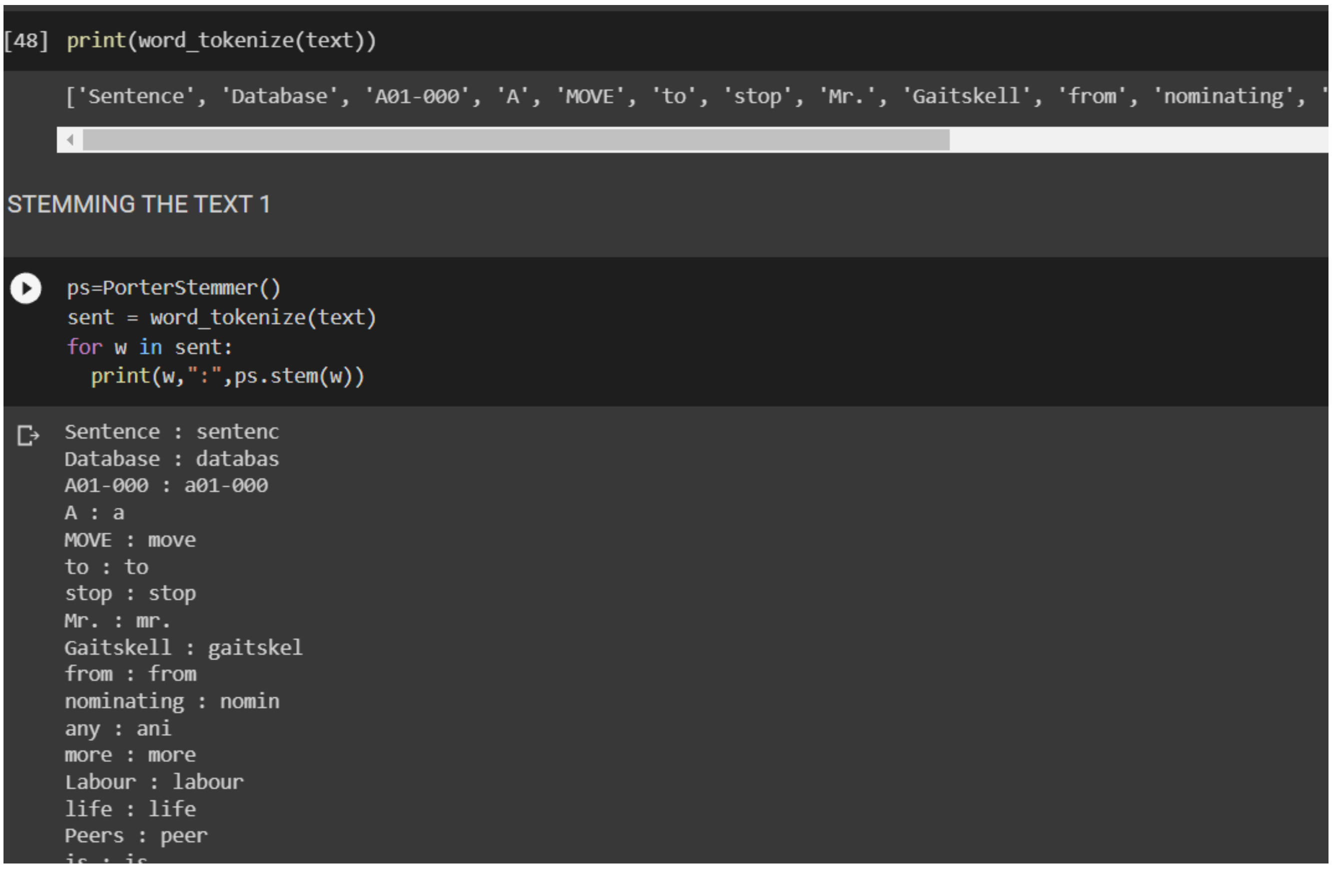
```
['Sentence', 'Database', 'A01-000', 'A', 'MOVE', 'stop', 'Mr.', 'Gaitske
'Sentence \n\nSentence Database A01-000\n\nA MOVE to stop Mr. Gaitskell
t has put down\na resolution on the subject and he is to be backed by Mr
stop Mr. Gaitskell from nominating any more Labour life Peers is to\nbe
to be backed by Mr. Will Griffiths, M P for\n\nManchester Exchange.\n\n
e Peers is to\nbe made at a meeting of Labour M Ps tomorrow. Mr. Michael
ster Exchange.\n\n \n\x0cA \n\nSentence Database A01-000\n\nA MOVE to st
```

TOKENIZATION AND STEMMING



After Removal of stop words, we have processed the text by tokenization the sentences and then used stemming process for further Pre-processing.

Tokenization is the process of dividing text into a set of meaningful pieces. These pieces are called tokens. Stemming is a process where words are reduced to a root by removing inflection through dropping unnecessary characters, usually a suffix



```
[48] print(word_tokenize(text))

     ['Sentence', 'Database', 'A01-000', 'A', 'MOVE', 'to', 'stop', 'Mr.', 'Gaitskell', 'from', 'nominating', '
```

STEMMING THE TEXT 1

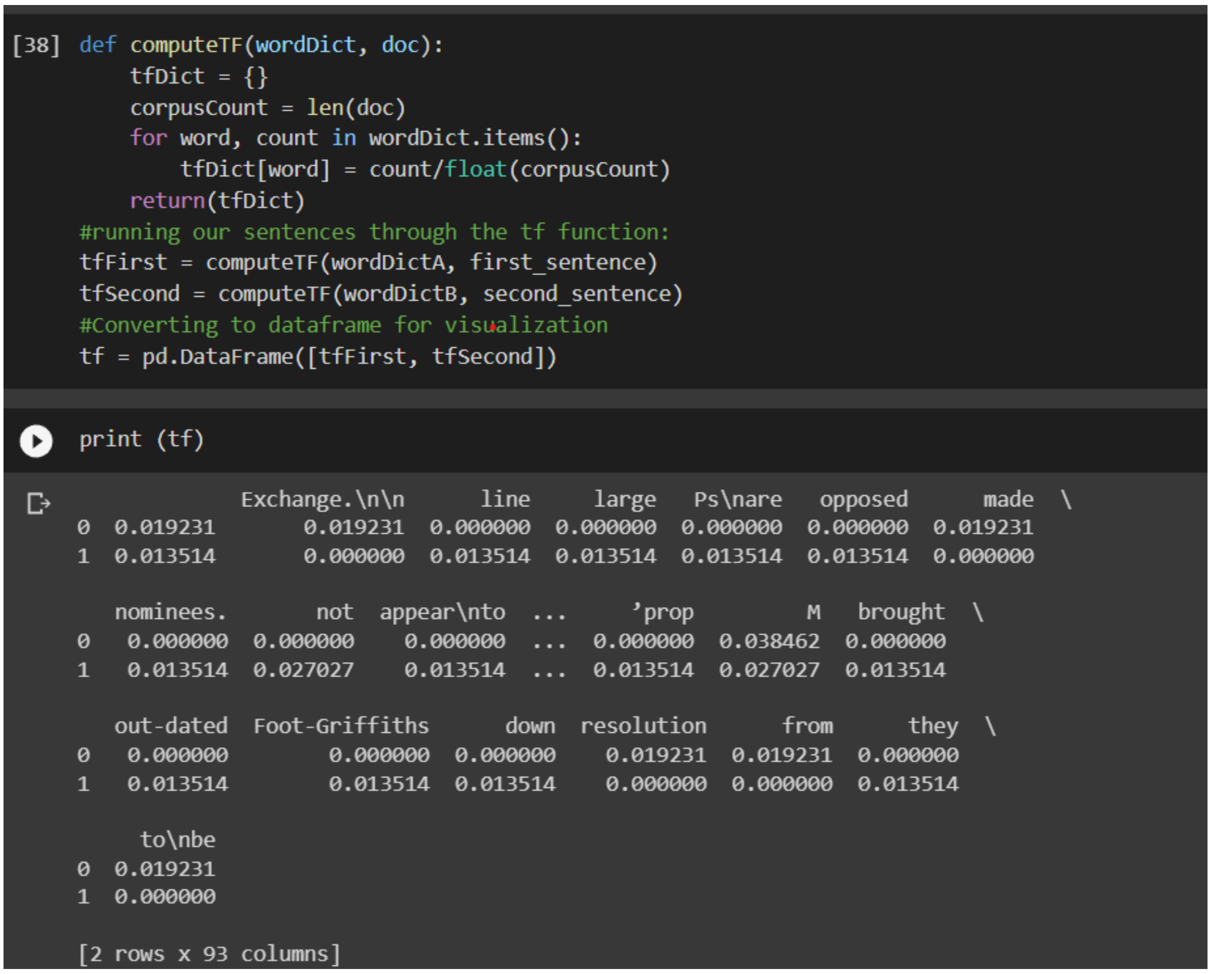
```
ps=PorterStemmer()
sent = word_tokenize(text)
for w in sent:
  print(w,":",ps.stem(w))
```

```
Sentence : sentenc
Database : databas
A01-000 : a01-000
A : a
MOVE : move
to : to
stop : stop
Mr. : mr.
Gaitskell : gaitskel
from : from
nominating : nomin
any : ani
more : more
Labour : labour
life : life
Peers : peer
is : is
```

FEATURE EXTRACTION TF AND IDF

TF-IDF stands for Term Frequency Inverse Document Frequency of records. It can be defined as the calculation of how relevant a word in a series or corpus is to a text. The meaning increases proportionally to the number of times in the text a word appears but is compensated by the word frequency in the corpus.

TF

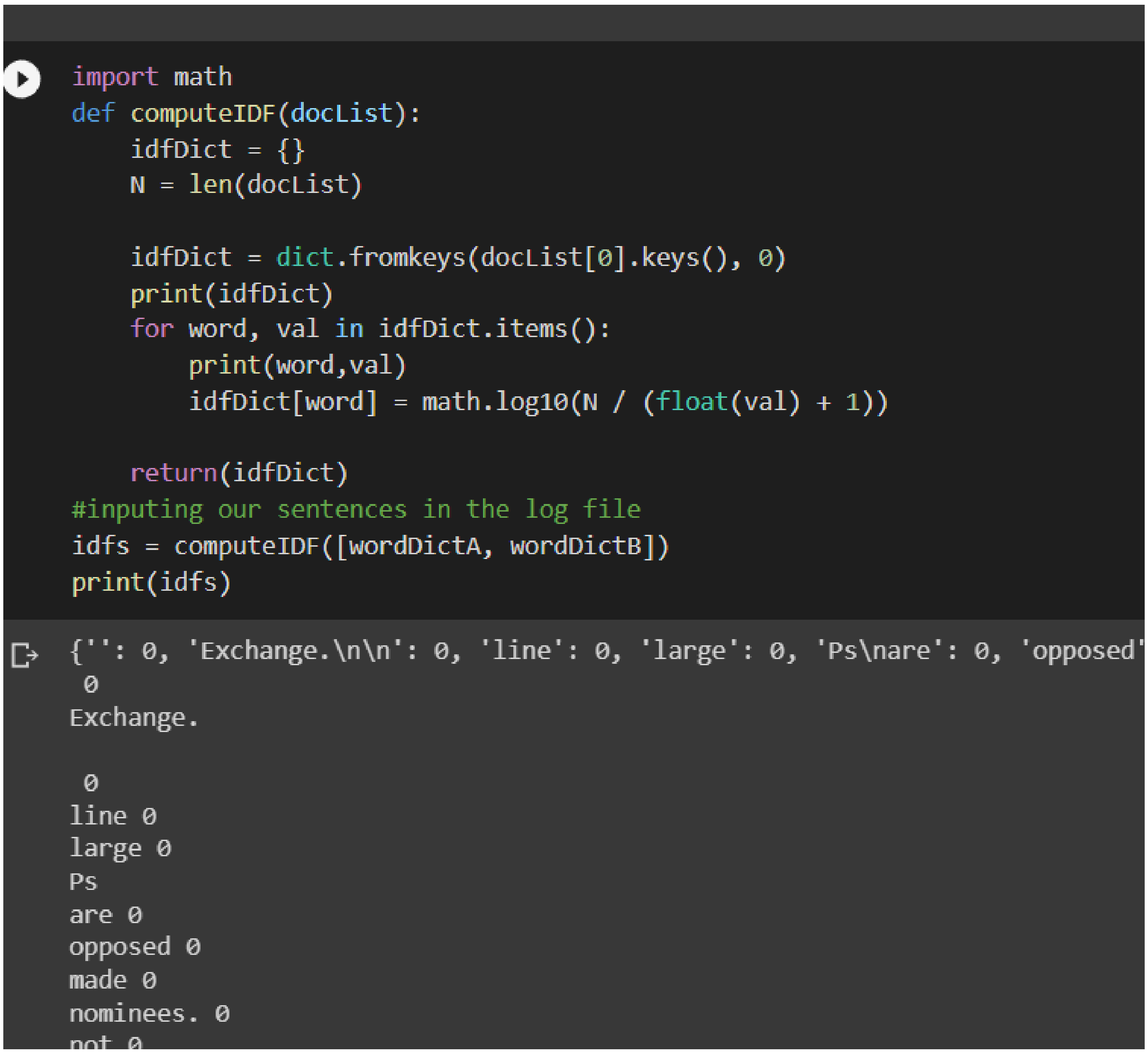


```
[38] def computeTF(wordDict, doc):
         tfDict = {}
         corpusCount = len(doc)
         for word, count in wordDict.items():
             tfDict[word] = count/float(corpusCount)
         return(tfDict)
     #running our sentences through the tf function:
     tfFirst = computeTF(wordDictA, first_sentence)
     tfSecond = computeTF(wordDictB, second_sentence)
     #Converting to dataframe for visualization
     tf = pd.DataFrame([tfFirst, tfSecond])
```

```
print (tf)
```

```
                Exchange.\n\n      line     large   Ps\nare   opposed      made  \
0  0.019231          0.019231  0.000000  0.000000  0.000000  0.000000  0.019231
1  0.013514          0.000000  0.013514  0.013514  0.013514  0.013514  0.000000

   nominees.       not  appear\nto  ...     ’prop         M   brought  \
0   0.000000  0.000000    0.000000  ...  0.000000  0.038462  0.000000
1   0.013514  0.027027    0.013514  ...  0.013514  0.027027  0.013514

   out-dated  Foot-Griffiths      down  resolution      from      they  \
0   0.000000        0.000000  0.000000    0.019231  0.019231  0.000000
1   0.013514        0.013514  0.013514    0.000000  0.000000  0.013514

     to\nbe
0  0.019231
1  0.000000

[2 rows x 93 columns]
```

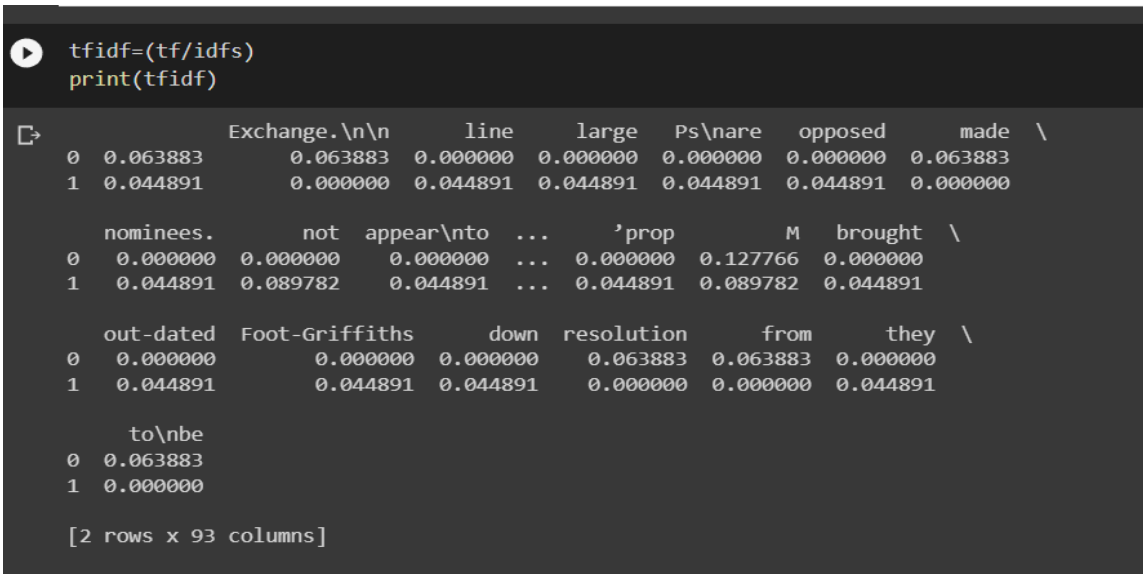
IDF



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

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

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

```
{'': 0, 'Exchange.\n\n': 0, 'line': 0, 'large': 0, 'Ps\nare': 0, 'opposed'
 0
Exchange.

 0
line 0
large 0
Ps
are 0
opposed 0
made 0
nominees. 0
not 0
```

TFIDF



```
tfidf=(tf/idfs)
print(tfidf)
```

```
              Exchange.\n\n      line     large   Ps\nare   opposed      made  \
0  0.063883        0.063883  0.000000  0.000000  0.000000  0.000000  0.063883
1  0.044891        0.000000  0.044891  0.044891  0.044891  0.044891  0.000000

   nominees.       not  appear\nto  ...     ’prop         M   brought  \
0   0.000000  0.000000    0.000000  ...  0.000000  0.127766  0.000000
1   0.044891  0.089782    0.044891  ...  0.044891  0.089782  0.044891

   out-dated  Foot-Griffiths      down  resolution      from      they  \
0   0.000000        0.000000  0.000000    0.063883  0.063883  0.000000
1   0.044891        0.044891  0.044891    0.000000  0.000000  0.044891

     to\nbe
0  0.063883
1  0.000000

[2 rows x 93 columns]
```