```
import nltk
nltk.download('punkt')

     [nltk_data] Downloading package punkt to /root/nltk_data...
     [nltk_data]   Unzipping tokenizers/punkt.zip.
     True
```

```
# create your own corpus. remove stopwords. tokenize the words. perform stemming.
```

```
from nltk.corpus.reader import WordListCorpusReader
```

```
x = WordListCorpusReader('/content/sample_data/', ['wordList.txt'])
temp = x.words()
x.words()
```

```
para= "After winning the Premier League and FA Cup, City emulated Manchester United's triple trophy haul in 1999 as they bec
```

```
from nltk.tokenize import word_tokenize
word_token = word_tokenize(para)
print(word_token)

     ['After', 'winning', 'the', 'Premier', 'League', 'and', 'FA', 'Cup', ',', 'City', 'emulated', 'Manchester', 'United', "'
```

```
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
print(stopwords.words('english'))

     ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'you
     [nltk_data] Downloading package stopwords to /root/nltk_data...
     [nltk_data]   Unzipping corpora/stopwords.zip.
```

```
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
```

```
ps = PorterStemmer()
for words in word_token:
  print(ps.stem(words))

     after
     win
     the
     premier
     leagu
     and
     fa
     cup
     ,
     citi
     emul
     manchest
     unit
     's
     tripl
     trophi
     haul
     in
     1999
     as
     they
     becam
     onli
     the
     second
     english
     club
     to
     achiev
     the
     feat
     after
     rodri
     's
     crisp
     68th-minut
     strike
     settl
     an
     attrit
```

```
    final
    .
```

```
rem_stopwords = [i for i in word_token if not i.lower() in stopWords]
print(rem_stopwords)
```

```
    ['winning', 'Premier', 'League', 'FA', 'Cup', ',', 'City', 'emulated', 'Manchester', 'United', "'s", 'triple', 'trophy',
```

```
psm = PorterStemmer()

for words in rem_stopwords:
  print(psm.stem(words))
```

```
    win
    premier
    leagu
    fa
    cup
    ,
    citi
    emul
    manchest
    unit
    's
    tripl
    trophi
    haul
    1999
    becam
    second
    english
    club
    achiev
    feat
    rodri
    's
    crisp
    68th-minut
    strike
    settl
    attrit
    final
    .
```

```
pst = PorterStemmer()
for words in more_filtered:
  print(words, ": ", pst.stem(words))
```

```
    general :  gener
    text :  text
    . :  .
    going :  go
    write :  write
    whatever :  whatev
    comes :  come
    mind :  mind
    . :  .
    would :  would
    much :  much
    like :  like
    go :  go
    back :  back
    home :  home
    already :  alreadi
    sick :  sick
    semester :  semest
    . :  .
    Man :  man
    City :  citi
    best :  best
    team :  team
    world :  world
    ! :  !
    ! :  !
```

```
#Lower casing
sentence = "Converting THE UpPer Case to Lower CAsE"
sentence = sentence.lower()
print(sentence)
```

```
    converting the upper case to lower case
```

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from nltk.corpus.reader import WordListCorpusReader
```

```
x = WordListCorpusReader('/content/sample_data', ['soda.txt'])
temp = x.words()
x.words()
```

```
    ['At age 29, President Biden became one of the youngest people ever elected to the United States Senate. Just weeks
    after his Senate election, tragedy struck the Biden family when his wife Neilia and daughter Naomi were killed, and
    sons Hunter and Beau were critically injured, in an auto accident.',
     'Biden was sworn into the U.S. Senate at his sons' hospital bedsides and began commuting from Wilmington to Washington
    every day, first by car, and then by train, in order to be with his family. He would continue to do so throughout his
    time in the Senate. ',
     'Biden married Jill Jacobs in 1977, and in 1980, their family was complete with the birth of Ashley Blazer Biden. A
    lifelong educator, Jill earned her doctorate in education and returned to teaching as an English professor at a
    community college in Virginia.',
     'Beau Biden, Attorney General of Delaware and Joe Biden's eldest son, passed away in 2015 after battling brain cancer
    with the same integrity, courage, and strength he demonstrated every day of his life. Beau's fight with cancer inspires
    the mission of President Biden's life — ending cancer as we know it.']
```

```
print(x)
```

```
    <WordListCorpusReader in '/content/sample_data'>
```

```
print(temp)
```

```
    ['At age 29, President Biden became one of the youngest people ever elected to the United States Senate. Just weeks afte
```

```
vec = CountVectorizer()
X = vec.fit_transform(temp)
```

```
df = pd.DataFrame(X.toarray(), columns = vec.get_feature_names_out())
df.head()
```

```
tfidf = TfidfVectorizer()
```

```
result = tfidf.fit_transform(temp)
result
```

```
    <4x118 sparse matrix of type '<class 'numpy.float64'>'
            with 153 stored elements in Compressed Sparse Row format>
```

```
print('\nidf values: ')
for el1, el2 in zip(tfidf.get_feature_names_out(), tfidf.idf_):
  print(el1, ':', el2)
```

```
    senate : 1.5108256237659907
    so : 1.916290731874155
    son : 1.916290731874155
    sons : 1.5108256237659907
    states : 1.916290731874155
    strength : 1.916290731874155
    struck : 1.916290731874155
    sworn : 1.916290731874155
    teaching : 1.916290731874155
    the : 1.0
    their : 1.916290731874155
    then : 1.916290731874155
    throughout : 1.916290731874155
    time : 1.916290731874155
    to : 1.2231435513142097
    tragedy : 1.916290731874155
    train : 1.916290731874155
    united : 1.916290731874155
    virginia : 1.916290731874155
    was : 1.5108256237659907
    washington : 1.916290731874155
    we : 1.916290731874155
    weeks : 1.916290731874155
    were : 1.916290731874155
    when : 1.916290731874155
    wife : 1.916290731874155
    wilmington : 1.916290731874155
    with : 1.2231435513142097
    would : 1.916290731874155
    youngest : 1.916290731874155
```

```
print("\nWord indices:")
print(tfidf.vocabulary_)
```

```
    Word indices:
    {'at': 11, 'age': 6, '29': 3, 'president': 84, 'biden': 21, 'became': 18, 'one': 80, 'of': 79, 'the': 97, 'youngest': 11
```

```
print("\n tf-idf values: ")
print(result)
```

```
print('\ntf-idf values in matrix form:')
print(temp)
print(result.toarray())
```

```
    tf-idf values in matrix form:
    ['At age 29, President Biden became one of the youngest people ever elected to the United States Senate. Just weeks afte
    [[0.         0.         0.         0.14887788 0.14887788 0.11737703
      0.14887788 0.11737703 0.23307196 0.         0.         0.09502682
      0.         0.14887788 0.         0.         0.         0.11737703
      0.14887788 0.         0.         0.1553813  0.         0.
      0.         0.         0.         0.         0.         0.
      0.         0.         0.         0.         0.14887788 0.14887788
      0.         0.         0.         0.         0.         0.
      0.         0.         0.         0.14887788 0.14887788 0.
      0.         0.14887788 0.         0.09502682 0.         0.
      0.         0.         0.         0.         0.19005364 0.
      0.14887788 0.07769065 0.14887788 0.         0.         0.
      0.         0.         0.         0.         0.14887788 0.14887788
      0.         0.         0.         0.         0.         0.14887788
      0.14887788 0.09502682 0.14887788 0.         0.         0.14887788
      0.11737703 0.         0.         0.         0.23475406 0.
      0.         0.11737703 0.14887788 0.         0.14887788 0.
      0.         0.23307196 0.         0.         0.         0.
      0.09502682 0.14887788 0.         0.14887788 0.         0.
      0.         0.         0.14887788 0.29775575 0.14887788 0.14887788
      0.         0.         0.         0.14887788]
     [0.         0.         0.         0.         0.         0.
      0.         0.         0.15973683 0.         0.         0.09769053
      0.         0.         0.         0.         0.1530511  0.
      0.         0.1530511  0.1530511  0.07986841 0.         0.
      0.         0.3061022  0.         0.1530511  0.         0.
      0.1530511  0.         0.1530511  0.         0.         0.
      0.12066724 0.         0.         0.1530511  0.         0.
      0.         0.         0.         0.         0.         0.
      0.         0.         0.12066724 0.09769053 0.         0.1530511
      0.1530511  0.         0.12066724 0.         0.2930716  0.1530511
      0.         0.15973683 0.         0.         0.         0.1530511
      0.         0.         0.         0.         0.         0.
      0.         0.         0.         0.         0.         0.
      0.         0.         0.         0.1530511  0.         0.
      0.         0.         0.         0.         0.24133449 0.1530511
      0.         0.12066724 0.         0.         0.         0.1530511
      0.         0.15973683 0.         0.1530511  0.1530511  0.1530511
      0.2930716  0.         0.1530511  0.         0.         0.12066724
      0.1530511  0.         0.         0.         0.         0.
```

```
  0.1530511  0.09769053 0.1530511  0.        ]
 [0.16504753 0.16504753 0.         0.         0.         0.
  0.         0.13012537 0.1722573  0.13012537 0.16504753 0.1053477
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.1722573  0.16504753 0.16504753
  0.         0.         0.         0.         0.16504753 0.16504753
  0.         0.16504753 0.         0.         0.         0.
  0.         0.         0.         0.         0.16504753 0.16504753
  0.16504753 0.16504753 0.         0.         0.         0.
  0.16504753 0.         0.         0.1053477  0.         0.
  0.         0.         0.         0.16504753 0.         0.
  0.         0.34451459 0.         0.         0.         0.
  0.         0.16504753 0.33009506 0.         0.         0.
  0.         0.         0.16504753 0.16504753 0.         0.
  0.         0.1053477  0.         0.         0.         0.
  0.         0.16504753 0.16504753 0.         0.         0.
```

```
print("Cosine Similarity")
```

```
    Cosine Similarity
```

```
from sklearn.metrics.pairwise import cosine_similarity


def compute_cosine_similarity(text1, text2):
  list_text = [text1, text2]

  vectorizer = TfidfVectorizer(stop_words='english')
  vectorizer.fit_transform(list_text)
  tfidf_text1, tfidf_text2 = vectorizer.transform([list_text[0]]), vectorizer.transform([list_text[1]])

  cs_score = cosine_similarity(tfidf_text1, tfidf_text2)

  return np.round(cs_score[0][0], 2)


cosine_similarity12 = compute_cosine_similarity(temp[0], temp[1])
cosine_similarity13 = compute_cosine_similarity(temp[0], temp[2])
cosine_similarity23 = compute_cosine_similarity(temp[1], temp[2])

print('The cosine similarity of sentence 1 and 2 is {}.'.format(cosine_similarity12))
print('The cosine similarity of sentence 1 and 3 is {}.'.format(cosine_similarity13))
print('The cosine similarity of sentence 2 and 3 is {}.'.format(cosine_similarity23))
```

```
    The cosine similarity of sentence 1 and 2 is 0.18.
    The cosine similarity of sentence 1 and 3 is 0.09.
    The cosine similarity of sentence 2 and 3 is 0.07.
```

```
def jaccard_similarity(x, y):
  intersection_cardinality = len(set.intersection(*[set(x), set(y)]))
  union_cardinality = len(set.union(*[set(x), set(y)]))
  return float(intersection_cardinality)/(union_cardinality)


jaccard_similarity(temp[0], temp[1])
jaccard_similarity(temp[0], temp[2])
jaccard_similarity(temp[1], temp[2])
```

```
    0.6
```

```
import nltk
nltk.download('punkt')

     [nltk_data] Downloading package punkt to /root/nltk_data...
     [nltk_data]   Unzipping tokenizers/punkt.zip.
     True


from nltk.corpus.reader import WordListCorpusReader


x1 = WordListCorpusReader('/content/sample_data/', ['1 GeorgeWashington.txt'])
temp1 = x1.words()
x1.words()

     ['George Washington (February 22, 1732 – December 14, 1799) was an American military officer, statesman, and Founding
     Father who served as the first president of the United States from 1789 to 1797. Appointed by the Second Continental
     Congress as commander of the Continental Army in June 1775, Washington led Patriot forces to victory in the American
     Revolutionary War and then served as president of the Constitutional Convention in 1787, which drafted and ratified the
     Constitution of the United States and established the American federal government. Washington has thus been called the
     "Father of his Country".']


x2 = WordListCorpusReader('/content/sample_data/', ['2 JohnAdams.txt'])
temp2 = x2.words()


x3 = WordListCorpusReader('/content/sample_data/', ['3 ThomasJefferson.txt'])
temp3 = x3.words()


x4 = WordListCorpusReader('/content/sample_data/', ['4 JamesMadison.txt'])
temp4 = x4.words()


x5 = WordListCorpusReader('/content/sample_data/', ['5 JamesMonroe.txt'])
temp5 = x5.words()


from nltk.tokenize import word_tokenize


word_token1 = word_tokenize(temp1[0])
print(word_token1)

     ['George', 'Washington', '(', 'February', '22', ',', '1732', '–', 'December', '14', ',', '1799', ')', 'was', 'an', 'Amer


word_token2 = word_tokenize(temp2[0])
print(word_token2)

     ['John', 'Adams', '(', 'October', '30', ',', '1735', '–', 'July', '4', ',', '1826', ')', 'was', 'an', 'American', 'state


word_token3 = word_tokenize(temp3[0])
print(word_token3)

     ['Thomas', 'Jefferson', '(', 'April', '13', ',', '1743', '[', 'b', ']', '–', 'July', '4', ',', '1826', ')', 'was', 'an',


word_token4 = word_tokenize(temp4[0])
print(word_token4)

     ['James', 'Madison', '(', 'March', '16', ',', '1751', '[', 'b', ']', '–', 'June', '28', ',', '1836', ')', 'was', 'an', '


word_token5 = word_tokenize(temp5[0])
print(word_token5)

     ['James', 'Monroe', '(', 'April', '28', ',', '1758', '–', 'July', '4', ',', '1831', ')', 'was', 'an', 'American', 'state

#REMOVAL OF STOPWORDS


import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
stop = stopwords.words('english')

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


for word in word_token1:
  if word in stop:
    word_token1.remove(word)
```

```
print(word_token1)

     ['George', 'Washington', '(', 'February', '22', ',', '1732', '–', 'December', '14', ',', '1799', ')', 'an', 'American',


for word in word_token2:
  if word in stop:
    word_token2.remove(word)


for word in word_token3:
  if word in stop:
    word_token3.remove(word)


for word in word_token4:
  if word in stop:
    word_token4.remove(word)


for word in word_token5:
  if word in stop:
    word_token5.remove(word)


from nltk.stem import PorterStemmer
ps = PorterStemmer()


punctuations = [',', '.', '!', '(', ')', '-', '_','&', '?']


!pip install unidecode
from unidecode import unidecode
import string

     Collecting unidecode
       Downloading Unidecode-1.3.6-py3-none-any.whl (235 kB)
          ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 235.9/235.9 kB 4.1 MB/s eta 0:00:00
     Installing collected packages: unidecode
     Successfully installed unidecode-1.3.6


def remove_unicode_punctuation(txt):
  normalised_text = unidecode(txt)

  cleaned_text = normalised_text.translate(txt.maketrans('', '', string.punctuation))

  return cleaned_text


inp_string = "Héllò! Résumé in the café."

cleanTxt = remove_unicode_punctuation(inp_string)
cleanTxt

     'Hello Resume in the cafe'


for word in range(0,len(word_token1)):
  word_token1[word] = remove_unicode_punctuation(word_token1[word])
word_token1


for word in range(0,len(word_token2)):
  word_token2[word] = remove_unicode_punctuation(word_token2[word])

for word in range(0,len(word_token3)):
  word_token3[word] = remove_unicode_punctuation(word_token3[word])

for word in range(0,len(word_token4)):
  word_token4[word] = remove_unicode_punctuation(word_token4[word])

for word in range(0,len(word_token5)):
  word_token5[word] = remove_unicode_punctuation(word_token5[word])


numeric = {"zero": 0, "one":1, "two":2, "three":3, "four": 4, "five":5, "six":6, "seven":7, "eight":8, "nine":9}


for word in word_token1:
  if word in numeric:
    word = numeric[word]
```

```
inp = ['one', 'two', 'zero']
for word in range(0,3):
  if inp[word] in numeric:
    inp[word] = numeric[inp[word]]

inp
```

```
    [1, 2, 0]
```

```
from nltk.stem import PorterStemmer
ps = PorterStemmer()
```

```
ps = PorterStemmer()
for word in range(0, len(word_token1)):
  word_token1[word] = ps.stem(word_token1[word])

word_token1
```

```
    ['georg',
     'washington',
     '(',
     'februari',
     '22',
     ',',
     '1732',
     '–',
     'decemb',
     '14',
     ',',
     '1799',
     ')',
     'an',
     'american',
     'militari',
     'offic',
     ',',
     'statesman',
     ',',
     'found',
     'father',
     'serv',
     'first',
     'presid',
     'unit',
     'state',
     '1789',
     '1797',
     '.',
     'appoint',
     'second',
     'contin',
     'congress',
     'command',
     'the',
     'contin',
     'armi',
     'june',
     '1775',
     ',',
     'washington',
     'led',
     'patriot',
     'forc',
     'victori',
     'the',
     'american',
     'revolutionari',
     'war',
     'then',
     'serv',
     'presid',
     'the',
     'constitut',
     'convent',
     '1787',
     ',',
import math
from collections import Counter

corpus = [ temp1[0], temp2[0], temp3[0], temp4[0], temp5[0]
    ]

tokenized_corpus = [doc.lower().split() for doc in corpus]

tf_corpus = []
for doc_tokens in tokenized_corpus:
    tf_doc = Counter(doc_tokens)
    tf_corpus.append(tf_doc)

unique_words = set(word for doc_tokens in tokenized_corpus for word in doc_tokens)
idf_corpus = {}
total_docs = len(corpus)

for word in unique_words:
    doc_count = sum(1 for doc_tokens in tokenized_corpus if word in doc_tokens)
    idf_corpus[word] = 1 + math.log(total_docs / (doc_count))

for i, (tf_doc, doc_tokens) in enumerate(zip(tf_corpus, tokenized_corpus)):
    print(f"Document {i + 1}: {doc_tokens}")
    print(list(tf_doc).sort)
    tfidf_doc = {word: tf_doc[word] * idf_corpus[word] for word in doc_tokens}
    print("TF-IDF:", tfidf_doc)
    print()
```

```
Document 1: ['george', 'washington', '(february', '22,', '1732', '–', 'december', '14,', '1799)', 'was', 'an', 'american
<built-in method sort of list object at 0x7eec934c60c0>
TF-IDF: {'george': 1.916290731874155, 'washington': 5.748872195622465, '(february': 2.6094379124341005, '22,': 2.6094379

Document 2: ['john', 'adams', '(october', '30,', '1735', '–', 'july', '4,', '1826)', 'was', 'an', 'american', 'statesman
<built-in method sort of list object at 0x7eec935c9980>
TF-IDF: {'john': 1.916290731874155, 'adams': 7.8283137373023015, '(october': 2.6094379124341005, '30,': 2.60943791243410

Document 3: ['thomas', 'jefferson', '(april', '13,', '1743[b]', '–', 'july', '4,', '1826)', 'was', 'an', 'american', 'st
<built-in method sort of list object at 0x7eec93576500>
TF-IDF: {'thomas': 1.916290731874155, 'jefferson': 7.8283137373023015, '(april': 1.916290731874155, '13,': 2.60943791243
```

```
def jaccard_similarity(x, y):
  intersection = len(set.intersection(*[set(x), set(y)]))
  union = len(set.union(*[set(x), set(y)]))
  return float(intersection)/union
```

```
jaccard_similarity(temp1[0], temp2[0])
```

```
0.8148148148148148
```

```
jaccard_similarity(temp1[0], temp3[0])
```

```
0.8148148148148148
```

```
jaccard_similarity(temp4[0], temp5[0])
```

```
0.6779661016949152
```

Write a program:

- To handle unicode characters- accented letter and some punctuation, eg: hyphen, dor, comma, etc.
- To perform lemmatization on a given sentence
- to implement the soundex algorithm
- to implement tf-idf (no built-in function)

```
!pip install unidecode
from unidecode import unidecode
import string
```

```
    Collecting unidecode
      Downloading Unidecode-1.3.6-py3-none-any.whl (235 kB)
         ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 235.9/235.9 kB 4.7 MB/s eta 0:00:00
    Installing collected packages: unidecode
    Successfully installed unidecode-1.3.6
```

```
def remove_unicode_punctuation(txt):
  normalised_text = unidecode(txt)

  cleaned_text = normalised_text.translate(txt.maketrans('', '', string.punctuation))

  return cleaned_text
```

```
inp_string = "Héllò, 世界! Surprised to see your résumé in the café."

cleaned_txt = remove_unicode_punctuation(inp_string)
cleaned_txt
```

```
    'Hello Shi Jie  Surprised to see your resume in the cafe'
```

PART II

```
!pip install nltk
import nltk
```

```
    Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (3.8.1)
    Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk) (8.1.7)
    Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk) (1.3.2)
    Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk) (2023.6.3)
    Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk) (4.66.1)
```

```
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
```

```
nltk.download("punkt")
nltk.download("wordnet")
```

```
    [nltk_data] Downloading package punkt to /root/nltk_data...
    [nltk_data]   Unzipping tokenizers/punkt.zip.
    [nltk_data] Downloading package wordnet to /root/nltk_data...
    True
```

```
def lemmatize(str):

  lemmatizer = WordNetLemmatizer()
  words = word_tokenize(str)
  lemmatized_words = [lemmatizer.lemmatize(i) for i in words]
  lemmatized_text = " ".join(lemmatized_words)

  return lemmatized_text
```

```
lemmatized_text = lemmatize(cleaned_txt)
lemmatized_text
```

```
    'Hello Shi Jie Surprised to see your resume in the cafe'
```

```
sample_txt = "dogs are running after the babies"
print(lemmatize(sample_txt))
```

```
    dog are running after the baby
```

```python
def soundex(word):

    soundex_mapping = {
        'b': '1', 'f': '1', 'p': '1', 'v': '1',
        'c': '2', 'g': '2', 'j': '2', 'k': '2', 'q': '2', 's': '2', 'x': '2', 'z': '2',
        'd': '3', 't': '3',
        'l': '4',
        'm': '5', 'n': '5',
        'r': '6'
    }

    word = word.lower()

    soundex_code = word[0]

    for char in word[1:]:
        if char in soundex_mapping:
            code = soundex_mapping[char]

            if code != soundex_code[-1]:
                soundex_code += code
    soundex_code = (soundex_code + '0000')[:4]

    return soundex_code



corpus = [
    "cats",
    "teacher",
    "outwest",
    "possible",
    "research"
]

soundex_encoded_corpus = [soundex(word) for word in corpus]

for i in range(len(corpus)):
    print(f"Soundex of the word {corpus[i]} is: {soundex_encoded_corpus[i]}")
```

```
Soundex of the word cats is: c320
Soundex of the word teacher is: t260
Soundex of the word outwest is: o323
Soundex of the word possible is: p214
Soundex of the word research is: r262
```

```python
import math
from collections import Counter

corpus = [
    "Manchester City won the Champions league in 2023",
    "They were the second team to win a treble in England",
    "Obamna Biden is the best president in the history of 2023",
    "This is the fourth and final document of this corpus!",
    "No this is the final document not the previous one, he is lying"
]

tokenized_corpus = [doc.lower().split() for doc in corpus]

tf_corpus = []
for doc_tokens in tokenized_corpus:
    tf_doc = Counter(doc_tokens)
    tf_corpus.append(tf_doc)

unique_words = set(word for doc_tokens in tokenized_corpus for word in doc_tokens)
idf_corpus = {}
total_docs = len(corpus)

for word in unique_words:
    doc_count = sum(1 for doc_tokens in tokenized_corpus if word in doc_tokens)
    idf_corpus[word] = 1 + math.log(total_docs / (doc_count))

for i, (tf_doc, doc_tokens) in enumerate(zip(tf_corpus, tokenized_corpus)):
    print(f"Document {i + 1}: {doc_tokens}")
    print("TF:", tf_doc)
    tfidf_doc = {word: tf_doc[word] * idf_corpus[word] for word in doc_tokens}
    print("TF-IDF:", tfidf_doc)
    print()
```

```
Document 1: ['manchester', 'city', 'won', 'the', 'champions', 'league', 'in', '2023']
TF: Counter({'manchester': 1, 'city': 1, 'won': 1, 'the': 1, 'champions': 1, 'league': 1, 'in': 1, '2023': 1})
```

```
TF-IDF: {'manchester': 2.6094379124341005, 'city': 2.6094379124341005, 'won': 2.6094379124341005, 'the': 1.0, 'champions

Document 2: ['they', 'were', 'the', 'second', 'team', 'to', 'win', 'a', 'treble', 'in', 'england']
TF: Counter({'they': 1, 'were': 1, 'the': 1, 'second': 1, 'team': 1, 'to': 1, 'win': 1, 'a': 1, 'treble': 1, 'in': 1, 'e
TF-IDF: {'they': 2.6094379124341005, 'were': 2.6094379124341005, 'the': 1.0, 'second': 2.6094379124341005, 'team': 2.609

Document 3: ['joe', 'biden', 'is', 'the', 'best', 'president', 'in', 'the', 'history', 'of', '2023']
TF: Counter({'the': 2, 'joe': 1, 'biden': 1, 'is': 1, 'best': 1, 'president': 1, 'in': 1, 'history': 1, 'of': 1, '2023':
TF-IDF: {'joe': 2.6094379124341005, 'biden': 2.6094379124341005, 'is': 1.5108256237659907, 'the': 2.0, 'best': 2.6094379

Document 4: ['this', 'is', 'the', 'fourth', 'and', 'final', 'document', 'of', 'this', 'corpus!']
TF: Counter({'this': 2, 'is': 1, 'the': 1, 'fourth': 1, 'and': 1, 'final': 1, 'document': 1, 'of': 1, 'corpus!': 1})
TF-IDF: {'this': 3.83258146374831, 'is': 1.5108256237659907, 'the': 1.0, 'fourth': 2.6094379124341005, 'and': 2.60943791

Document 5: ['no', 'this', 'is', 'the', 'final', 'document', 'not', 'the', 'previous', 'one']
TF: Counter({'the': 2, 'no': 1, 'this': 1, 'is': 1, 'final': 1, 'document': 1, 'not': 1, 'previous': 1, 'one': 1})
TF-IDF: {'no': 2.6094379124341005, 'this': 1.916290731874155, 'is': 1.5108256237659907, 'the': 2.0, 'final': 1.916290731
```

```
import requests
from bs4 import BeautifulSoup
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
```

```
def preprocess_text(url):

    response = requests.get(url)
    soup = BeautifulSoup(response.text, "html.parser")
    text = soup.get_text()

    #Tokenization
    tokens = word_tokenize(text)

    #Removal of stopwords and handling whitespace
    stop_words = set(stopwords.words("english"))
    tokens = [word.lower() for word in tokens if word.isalnum() and word.lower() not in stop_words]

    #Converting text to lowercase
    tokens = [word.lower() for word in tokens]

    #Lemmatization
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(word) for word in tokens]


    preprocessed_text = " ".join(tokens)

    return preprocessed_text

print(preprocess_text('https://www.amazon.in/'))
print(preprocess_text('https://www.iitb.ac.in/'))
print(preprocess_text('https://www.google.co.in/'))
print(preprocess_text('https://www.wikipedia.org/'))
print(preprocess_text('https://www.isro.gov.in/'))

text1=preprocess_text('https://www.amazon.in/')
text2=preprocess_text('https://www.iitb.ac.in/')
text3=preprocess_text('https://www.google.co.in/')
text4=preprocess_text('https://www.wikipedia.org/')
text5=preprocess_text('https://www.isro.gov.in/')
```

```
online shopping site india shop online mobile book watch shoe skip main content hello select address select department w
indian institute technology bombay iit bombay skip main content english indian institute technology bombay menu homeabou
googlesearch image map play youtube news gmail drive web history setting sign advanced searchadvertisingbusiness solutio
wikipedia wikipedia free encyclopedia english 6 715 article 1 387 1 892 ð 1 938 deutsch 2 836 artikel 2 553 article ital
indian space research organisation flash news spacecraft escaped sphere earth influence way lagrange point 1 l1 flash ne
```

```
from collections import Counter


frequency1=[Counter(text1.split())]
print(frequency1)
frequency2=[Counter(text2.split())]
print(frequency2)
frequency3=[Counter(text3.split())]
print(frequency3)
frequency4=[Counter(text4.split())]
print(frequency4)
frequency5=[Counter(text5.split())]
print(frequency5)
```

```
[Counter({'character': 2, 'see': 2, 'make': 2, 'sure': 2, 'image': 2, 'enter': 1, 'sorry': 1, 'need': 1, 'robot': 1, 'be
[Counter({'iit': 32, 'bombay': 26, 'research': 19, 'iitb': 10, 'technology': 9, 'cell': 9, 'new': 9, 'dual': 8, 'degree'
[Counter({'googlesearch': 1, 'image': 1, 'map': 1, 'play': 1, 'youtube': 1, 'news': 1, 'gmail': 1, 'drive': 1, 'web': 1,
[Counter({'free': 12, 'wikipedia': 9, '1': 9, 'article': 8, 'english': 5, 'ù': 5, 'bahasa': 5, 'ð': 4, 'basa': 4, 'donat
[Counter({'isro': 43, 'space': 31, 'test': 18, 'gaganyaan': 18, 'mission': 18, 'programme': 13, 'system': 11, 'engine':
```

```
zipfian1=[]
for c in frequency1:
  freq={word:1/(rank+1)for rank,(word,_)in enumerate(c.most_common())}
  zipfian1.append(freq)

print(zipfian1)
zipfian2=[]
for c in frequency2:
  freq={word:1/(rank+1)for rank,(word,_)in enumerate(c.most_common())}
  zipfian2.append(freq)

print(zipfian2)
zipfian3=[]
for c in frequency3:
  freq={word:1/(rank+1)for rank,(word,_)in enumerate(c.most_common())}
  zipfian3.append(freq)

print(zipfian3)
zipfian4=[]
for c in frequency4:
  freq={word:1/(rank+1)for rank,(word,_)in enumerate(c.most_common())}
  zipfian4.append(freq)

print(zipfian4)
zipfian5=[]
for c in frequency5:
  freq={word:1/(rank+1)for rank,(word,_)in enumerate(c.most_common())}
  zipfian5.append(freq)

print(zipfian5)
```

```
[{'character': 1.0, 'see': 0.5, 'make': 0.3333333333333333, 'sure': 0.25, 'image': 0.2, 'enter': 0.16666666666666666, 's
[{'iit': 1.0, 'bombay': 0.5, 'research': 0.3333333333333333, 'iitb': 0.25, 'technology': 0.2, 'cell': 0.1666666666666666
[{'googlesearch': 1.0, 'image': 0.5, 'map': 0.3333333333333333, 'play': 0.25, 'youtube': 0.2, 'news': 0.1666666666666666
[{'free': 1.0, 'wikipedia': 0.5, '1': 0.3333333333333333, 'article': 0.25, 'english': 0.2, 'ù': 0.16666666666666666, 'ba
[{'isro': 1.0, 'space': 0.5, 'test': 0.3333333333333333, 'gaganyaan': 0.25, 'mission': 0.2, 'programme': 0.1666666666666
```

```
diffs1=[]
for actual,freq in zip(frequency1,zipfian1):
  diff={word: actual[word]-freq.get(word,0)for word in actual}
  diffs1.append(diff)
print(diffs1)

diffs2=[]
for actual,freq in zip(frequency2,zipfian2):
  diff={word: actual[word]-freq.get(word,0)for word in actual}
  diffs2.append(diff)
print(diffs2)

diffs3=[]
for actual,freq in zip(frequency3,zipfian3):
  diff={word: actual[word]-freq.get(word,0)for word in actual}
  diffs3.append(diff)
print(diffs3)

diffs4=[]
for actual,freq in zip(frequency4,zipfian4):
  diff={word: actual[word]-freq.get(word,0)for word in actual}
  diffs4.append(diff)
print(diffs4)

diffs5=[]
for actual,freq in zip(frequency5,zipfian5):
  diff={word: actual[word]-freq.get(word,0)for word in actual}
  diffs5.append(diff)
print(diffs5)
```

```
[{'e': 29.0, 'n': 9.9, 't': 13.75, 'r': 15.666666666666666, ' ': 29.5, 'c': 10.875, 'h': 2.9444444444444446, 'a': 13.8,
[{'i': 737.6666666666666, 'n': 535.8333333333334, 'd': 265.9166666666667, 'a': 679.75, ' ': 929.0, 's': 431.875, 't': 61
[{'g': 6.857142857142857, 'o': 6.875, 'l': 3.9166666666666665, 'e': 12.5, 's': 11.666666666666666, 'a': 10.75, 'r': 6.88
[{'w': 47.95, 'i': 254.66666666666666, 'k': 89.9090909090909, 'p': 60.9375, 'e': 214.75, 'd': 73.93333333333334, 'a': 31
[{'i': 518.75, 'n': 494.8, 'd': 170.93333333333334, 'a': 596.6666666666666, ' ': 911.0, 's': 490.8333333333333, 'p': 185
```

```
percentage1=[]
for differ in diffs1:
  total=sum(differ.values())
  percent_diff={word: (value/total)*100 for word,value in differ.items()}
  percentage1.append(percent_diff)

print(percentage1)

percentage2=[]
for differ in diffs2:
  total=sum(differ.values())
  percent_diff={word: (value/total)*100 for word,value in differ.items()}
  percentage2.append(percent_diff)

print(percentage2)

percentage3=[]
for differ in diffs3:
  total=sum(differ.values())
  percent_diff={word: (value/total)*100 for word,value in differ.items()}
  percentage3.append(percent_diff)

print(percentage3)

percentage4=[]
for differ in diffs4:
  total=sum(differ.values())
  percent_diff={word: (value/total)*100 for word,value in differ.items()}
  percentage4.append(percent_diff)

print(percentage4)

percentage5=[]
for differ in diffs5:
  total=sum(differ.values())
  percent_diff={word: (value/total)*100 for word,value in differ.items()}
  percentage5.append(percent_diff)

print(percentage5)
```

```
    [{'e': 14.337576159920149, 'n': 4.894551861489982, 't': 6.797988696513864, 'r': 7.745587120876403, ' ': 14.5847757488842
    [{'i': 8.894106346623213, 'n': 6.460585608764941, 'd': 3.206178643478838, 'a': 8.19580043712218, ' ': 11.20102773973741,
    [{'g': 4.855506731476846, 'o': 4.868151280256734, 'l': 2.7733710323886847, 'e': 8.851184145921334, 's': 8.26110520285991
    [{'w': 1.6125554256469146, 'i': 8.564423671840405, 'k': 3.023636962678544, 'p': 2.049324217942833, 'e': 7.22202873112981
    [{'i': 7.4161055807029985, 'n': 7.073713814615602, 'd': 2.4436812477330103, 'a': 8.530010595635899, ' ': 13.023753607750
```

```
# import scrapy
from bs4 import BeautifulSoup
import re
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import requests


# URL=input("Enter the URL:")
# URL="https://medium.com/data-science-bootcamp/tf-idf-basics-of-information-retrieval-48de122b2a4c"
URL="https://www.indiacelebrating.com/essay/diwali-essay/"
req = requests.get(URL)
print(req.content)
```

```
    b'<!DOCTYPE html>\r\n<html lang="en-US">\r\n<head>\r\n<meta charset="UTF-8">\r\n<meta name="viewport" content="width=dev
```

```
# URL = "http://www.values.com/inspirational-quotes"
# req = requests.get(URL)

soup = BeautifulSoup(req.content, 'html.parser') # If this line causes an error, run 'pip install html5lib' or install html5
print(soup.prettify())
```

```
          <script async="" id="google_gtagjs-js" src="https://www.googletagmanager.com/gtag/js?id=G-TESBSTS4LC">
          </script>
          <script id="google_gtagjs-js-after">
           window.dataLayer = window.dataLayer || [];function gtag(){dataLayer.push(arguments);}
    gtag("js", new Date());
    gtag("set", "developer_id.dZTNiMT", true);
    gtag("config", "G-TESBSTS4LC");
          </script>
          <!-- End Google Analytics snippet added by Site Kit -->
          <link href="https://www.indiacelebrating.com/wp-json/" rel="https://api.w.org/"/>
          <link href="https://www.indiacelebrating.com/wp-json/wp/v2/posts/7958" rel="alternate" type="application/json"/>
          <link href="https://www.indiacelebrating.com/xmlrpc.php?rsd" rel="EditURI" title="RSD" type="application/rsd+xml"/
          <link href="https://www.indiacelebrating.com/wp-includes/wlwmanifest.xml" rel="wlwmanifest" type="application/wlwm
          <meta content="WordPress 6.2.2" name="generator"/>
          <link href="https://www.indiacelebrating.com/?p=7958" rel="shortlink"/>
          <link href="https://www.indiacelebrating.com/wp-json/oembed/1.0/embed?url=https%3A%2F%2Fwww.indiacelebrating.com%2
          <link href="https://www.indiacelebrating.com/wp-json/oembed/1.0/embed?url=https%3A%2F%2Fwww.indiacelebrating.com%2
          <meta content="Site Kit by Google 1.101.0" name="generator"/>
          <link href="https://www.indiacelebrating.com/xmlrpc.php" rel="pingback"/>
          <!-- Google AdSense snippet added by Site Kit -->
          <meta content="ca-host-pub-2644536267352236" name="google-adsense-platform-account"/>
          <meta content="sitekit.withgoogle.com" name="google-adsense-platform-domain"/>
          <!-- End Google AdSense snippet added by Site Kit -->
          <meta content="Elementor 3.13.3; features: e_dom_optimization, e_optimized_assets_loading, e_optimized_css_loading
          <!-- Google AdSense snippet added by Site Kit -->
          <script async="" crossorigin="anonymous" src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js?clien
          </script>
          <!-- End Google AdSense snippet added by Site Kit -->
          <script>
           if(screen.width<769){
                    window.location = "https://www.indiacelebrating.com/essay/diwali-essay/amp/";
           }
          </script>
          <link href="https://www.indiacelebrating.com/wp-content/uploads/cropped-i-32x32.png" rel="icon" sizes="32x32">
           <link href="https://www.indiacelebrating.com/wp-content/uploads/cropped-i-192x192.png" rel="icon" sizes="192x192"
           <link href="https://www.indiacelebrating.com/wp-content/uploads/cropped-i-180x180.png" rel="apple-touch-icon"/>
           <meta content="https://www.indiacelebrating.com/wp-content/uploads/cropped-i-270x270.png" name="msapplication-Til
            <script async="" crossorigin="anonymous" src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js?cli
            </script>
            <!-- Google tag (gtag.js) -->
            <script async="" src="https://www.googletagmanager.com/gtag/js?id=UA-163213831-1">
            </script>
            <script>
             window.dataLayer = window.dataLayer || [];
      function gtag(){dataLayer.push(arguments);}
      gtag('js', new Date());

      gtag('config', 'UA-163213831-1');
            </script>
           </meta>
          </link>
         </link>
        </link>
       </meta>
      </meta>
     </head>
     <body class="post-template-default single single-post postid-7958 single-format-standard wp-custom-logo group-blog ast-
      <svg focusable="false" height="0" role="none" style="visibility: hidden; position: absolute; left: -9999px; overflow:
       <defs>
```

```
print(soup.title)
print(soup.title.name)
print(soup.title.parent.name)
```

```
    <title>Long and Short Essay on Diwali for Children and Students</title>
    title
    meta
```

```
s=soup.find('article',class_="post-7958 post type-post status-publish format-standard hentry category-essay ast-article-sing
content=s.find_all("p")
print(content)
```

```
    [<p>Diwali also called “Deepawali” is a major Hindu festival of India. The festival is celebrated with unequal zeal and
```

```
lines=s.find_all("p")
final_text=[]
for i in lines:
    print(i.text)
    final_text.append(i.text)
```

```
    Diwali also called “Deepawali” is a major Hindu festival of India. The festival is celebrated with unequal zeal and plea
    Hindus believe that his return was welcomed by the people of Ayodhya by lighting up the streets and houses by small eart
    Speech on Diwali for School Students | Speech on Diwali for Teachers | Paragraph on Diwali
    Diwali is a religious Hindu festival, celebrated as festival of lights by lighting lamps everywhere at homes, streets, s
    People of Hindu religion wait very eagerly for this special festival of Diwali. It is the most important and favorite fe
    Use following long and short essay on Diwali to make your kids smart enough at home or school and motivate them to know
    You can select anyone of these Diwali essay according to your need:
    Diwali is one of the main festivals of Hindus. The preparation for Diwali celebration begins weeks before the festival.

    People shop for new clothes, home decor items and gifts for their loved ones on this festival. The markets are flooded w
    On the day of Diwali, people light up their houses with diyas, candles and lights. They also make rangoli and decorate t
    Also known as the festival of lights, Diwali is all about worshiping the deities, burning crackers, having sweets and ma

    Introduction
    Diwali is also known as Deepawali meaning a row of diyas. The festival is celebrated with great zeal throughout India. I
    Festival of Lights
    Lighting diyas is one of the main rituals of this Hindu festival. People buy beautiful earthenware diyas each year and i
    The houses, marketplaces, offices, temples and all the other places are illuminated with lights on this day. Candles, la
    Rangolis are made and diyas are placed in between these beautiful creations of art to enhance their look.

    Exchange of Gifts
    Exchanging gifts is one of the main rituals of the Diwali festival. People visit their colleagues, neighbours, relatives
    While exchanging sweets and boxes of dry fruit was common in the earlier times, these days people look for unique and in
    People also purchase gifts for their employees and house helps. Many people also visit orphanages and old age homes and
    Conclusion
    People await Diwali all year long and the preparations for its celebration begin almost a month before the festival. Peo

    Introduction
    As per the Hindu calendar, Diwali falls on the new moon (amavasya) during the Kartik month. This is considered to be one

    Cleaning and Decoration
    Diwali celebration begins with the cleaning of the houses and work places. From washing curtains to cleaning the fans, f
    People also shop for various home decor items to redecorate their places. The houses are decorated with diyas, lights, l
    Sharing the Joy
    People visit their relatives, neighbours and friends. They exchange gifts and spend time with each other. Many people ho
    Many residential societies organize Diwali parties to celebrate the occasion. It is a great way to rejoice in the festiv
    Worshipping the Deities
    Goddess Lakshmi and Lord Ganesha are worshipped during the evening hours. People wear new clothes and offer prayers to t
    Burning of Fire Crackers and Increasing Pollution
    Fire crackers are also burnt as a part of Diwali celebrations. Large numbers of crackers are burnt on this day each year
    Diwali without fire crackers would be much more beautiful. The newer generations must be sensitized about the harmful ef
    Conclusion
    Diwali, also known as the festival of lights, is a mark of the Hindu tradition. It is celebrated with joy and enthusiasm

    Introduction
    Diwali falls sometime between the mid of October and mid of November. It is one of the main festivals of Hindus. The fes
    Why Do we Celebrate Diwali?
    While it is largely believed that Diwali is celebrated to rejoice the return of Lord Rama to Ayodhya, many other folklor
    The Return of Lord Rama
    It is believed that on this day, Lord Rama returned to his hometown Ayodhya after staying in exile for fourteen long yea
    The entire town was illuminated with diyas. Sweets were distributed and people made merry. This is how we continue to ce
    The Harvest Festival
    In some parts of the country, Diwali is considered to be a harvest festival. This is because it is the time when rice is
    The Legend of Lord Vishnu and Goddess Lakshmi
    It is said that King Bali had imprisoned Goddess Lakshmi. It was on this day that Lord Vishnu disguised himself and set
    The Birth of Goddess Lakshmi
    It is said that Goddess Lakshmi was born on the new moon of the Kartik month. Thus, in certain regions, Diwali is celebr
    The ritual of worshipping Goddess Lakshmi and Lord Ganesha is followed in every Hindu household on the day of Diwali.
```

```
# import nltk
# from nltk.tokenize import word_tokenize
# from nltk.corpus import stopwords
# from nltk.stem import PorterStemmer
import string
sentences = final_text
```

```
stop_words = set(stopwords.words('english'))
stemmer = PorterStemmer()

preprocessed_sentences = []

for sentence in sentences:
    # Tokenization
    tokens = word_tokenize(sentence)

    # Remove stopwords and apply stemming
    filtered_tokens = [stemmer.stem(word.lower()) for word in tokens if word.lower() not in stop_words]

    # Normalization (remove punctuation)
    normalized_tokens = [word for word in filtered_tokens if word not in string.punctuation]

    # Combine the normalized tokens back into a sentence
    preprocessed_sentence = " ".join(normalized_tokens)

    preprocessed_sentences.append(preprocessed_sentence)

# print(preprocessed_sentences)
for i in preprocessed_sentences:
    print(i)
```

```
ritual worship goddess lakshmi lord ganesha follow everi hindu household day diwali
matter reason diwali celebr immens enthusiasm across india well countri clean hous shop new cloth sweet gift decor hous
conclus
diwali bring us closer near dear one peopl age group await festiv look forward celebr love one everi member famili take

introduct
diwali time meet greet love one prepar delici sweet wear new cloth redecor hous worship goddess lakshmi also time burn f
diwali celebr
diwali celebr india sinc ancient time day celebr victori light dark per hindu mytholog day lord rama return kingdom ayod
effigi ravana burnt across india dussehra year mark victori good evil diwali fall twenti day later hous marketplac illum
peopl visit exchang gift part diwali celebr mani peopl host hous parti day great time bond rel friend mani offic residen
children especi look forward burn fire cracker day gather around rejoic festiv burn differ kind cracker
diwali pollut matter concern
diwali auspici day entir atmospher fill air festiv joy around time howev eventu fill pollut fire cracker burnt day compl
burn cracker pollut air also caus nois pollut particularli disturb sick elderli peopl small kid student anim
eco-friendli diwali good idea
high time must behav respons citizen stop burn cracker celebr diwali occas matter must celebr eco-friendli diwali
must say cracker advis around us parent must take respons tell kid neg repercuss burn cracker kid must also sensit schoo
apart measur peopl take end import put check sale fire cracker govern must interven product sale fire cracker must ban l
conclus
diwali sacr festiv must maintain sanctiti celebr right way refrain burn cracker owe harm effect environ ultim impact lif

introduct
diwali signific hindu festiv celebr india autumn season everi year spiritu signific festiv indic victori light dark five
peopl worship god ganesha goddess lakshmi get wealth prosper life perform puja main diwali lot ritual puja get involv fi
celebr diwali famili without cracker
diwali favorit festiv year celebr lot enthusiasm famili member friend diwali call festiv light celebr light lot diya can
famili member spend day time prepar hous clean decor etc welcom festiv grand even parti neighbor famili member friend ge
peopl go home take job offic work student also book train around three month ago easili go home diwali festiv everyon wa
howev prohibit doctor got outsid enjoy firecrack especi peopl suffer lung heart diseas hypertens diabet etc peopl knock
signific diwali
diwali festiv celebr peopl great revelri lot fun frolic activ becom happiest holiday indian peopl year celebr signific p
mani ancient stori legend myth celebr festiv girl women home shop make rangoli creativ pattern floor near door walkway h
spiritu signific festiv symbol victori light dark victori good evil celebr honor goddess wealth lakshmi god wisdom ganes
peopl celebr rememb return pandava kingdom 12 year vanva one year agyatava accord hindu epic mahabharata also believ sta
pollut diwali
togeth diwali celebr indirect increas environment pollut world burst variou type firecrack festiv firecrack danger relea
conclus
now-a-day campaign run govern celebr pollut free diwali countri school variou organ also organ variou demonstr prior cel
air water pollut also caus decay remnant firework delug garbag like empti bottl paper use light rocket gift wrapper dri

relat inform
slogan diwali
paragraph diwali
inform diwali festiv
dussehra essay
essay festiv india
essay holi
essay ganesh chaturthi
diwali kid
essay pollut due diwali

also see
essay dhantera
essay bhai dooj
essay govardhan puja
essay dev deepawali
essay kali puja
```

```
import requests
from bs4 import BeautifulSoup

URL = "https://insights.blackcoffer.com/the-rise-of-the-ott-platform-and-its-impact-on-the-entertainment-industry-by-2040/"
r = requests.get(URL)

soup = BeautifulSoup(r.content, 'html5lib') # If this line causes an error, run 'pip install html5lib' or install html5lib
# print(soup.prettify())
text1 = soup.find('div', class_ = "td-post-content tagdiv-type").text
text1
```

```
'\n        \n“Entertainment is giving people what they want.” – Howard Hawks\n\n\n\nEntertainment, in its essence, is t
he art of captivating and delighting people, whisking them away from the mundane and into a world of imagination and em
otion. It comes in various forms, from the tales told around ancient campfires to the dazzling spectacles of modern cin
ema. Traditionally, we relied on television, radio, and theatres to satiate our thirst for entertainment. However, a ne
w dawn has emerged with the advent of Over-The-Top (OTT) platforms, which deliver a wide array of digital content direc
tly to consumers over the internet, transforming the way we consume content. These platforms, accessible at the tap of
a screen, are redefining the entertainment landscape, and their impact is both revolutionary and intriguing. As we proj
ect into the future, it is evident that the impact of OTT platforms on the entertainment industry will be profound by 2
```

```
URL = "https://insights.blackcoffer.com/how-does-metaverse-work-in-the-financial-sector-2/"
r = requests.get(URL)

soup = BeautifulSoup(r.content, 'html5lib') # If this line causes an error, run 'pip install html5lib' or install html5lib
# print(soup.prettify())
text2 = soup.find('div', class_ = "td-post-content tagdiv-type").text
text2
```

```
'\n        \nWhat is Metaverse? \n\n\nMetaverse is an online, 3D universe that combines multiple different virtual spac
es. It is just a concept for now, but the way it is growing it is difficult to say that it would remain a concept in th
e coming future. Metaverse would include all the things that are available in reality. Everything would be accessible i
n this 3D universe.\n\n\nMetaverse is the easiest way to create your own Augmented Reality Experiences.\n\n\nHow to acc
ess Metaverse?\n\n\nA major discussion is ongoing if the Metaverse would be accessible without paying any real money or
it would be only accessible after some amount is paid to the owners of the Metaverse.\n\n\nHowever, currently, the vide
o game companies and similar Metaverse companies are not charging their customers a penny, so we are still inclined tha
t the judgment shall be towards not paying and free access to the online 3D universe.\n\n\nWhy is Metaverse gaining pop
```

```
URL = "https://insights.blackcoffer.com/the-impact-of-the-metaverse-on-financial-services/"
r = requests.get(URL)

soup = BeautifulSoup(r.content, 'html5lib') # If this line causes an error, run 'pip install html5lib' or install html5lib
# print(soup.prettify())
text3 = soup.find('div', class_ = "td-post-content tagdiv-type").text
text3
```

```
'\n        \nA Metaverse is a virtual world that combines thoughts from the I
nternet, augmented reality, virtual reality, artificial intelligence, and dif
ferent advances.\n\n\nIt is viewed as the association of the relative multitu
de of advances in the cutting edge in innovation, where a virtual world is po
ssessed by real people who use programming tools, augmented reality, artifici
al intelligence, human/hardware interfaces, and other modern technologies to
connect.\n\n\n\xa0When joined with cryptocurrencies, the Metaverse idea takes
into consideration brilliant properties and digital entities to be connected
```

```
URL = "https://insights.blackcoffer.com/how-metaverse-and-vr-can-reform-work-culture/"
r = requests.get(URL)

soup = BeautifulSoup(r.content, 'html5lib') # If this line causes an error, run 'pip install html5lib' or install html5lib
# print(soup.prettify())
text4 = soup.find('div', class_ = "td-post-content tagdiv-type").text
text4
```

```
'\n        \nCovid-19 has been declared a pandemic by the World Health Organi
zation for more than a year. Lockdowns have affected hundreds of millions of
people. Many people have made the abrupt transition to working from home, and
millions have lost their jobs. The future appears to be uncertain. We have no
idea when or if our societies will be able to return to normalcy – or what ki
nd of scars the pandemic will leave. Since then, organizations have been tryi
ng to innovate and make the work environment flexible and immersive.\n\n\nOne
of those recent innovations is Metaverse VR. Backed by none other than the pi
```

```
URL = "https://insights.blackcoffer.com/how-metaverse-will-change-your-life/"
r = requests.get(URL)

soup = BeautifulSoup(r.content, 'html5lib') # If this line causes an error, run 'pip install html5lib' or install html5lib
# print(soup.prettify())
text5 = soup.find('div', class_ = "td-post-content tagdiv-type").text
text5
```

```
'\n        \nEver since the stone age, humans have been evolving in one way o
r the other. In the recent past, this wave of evolution is looking stronger t
han ever. If you look at the major turning points in human history, the inven
tion of the internet happens to be one of the most significant ones. The inte
rnet has largely revolutionized humanity which makes us question "what the fu
```

```
import re
docs = [text1,text2,text3,text4,text5]
tokened_docs = []
def tokenization(text):
   tokens = text.split(' ')
   text_t = " ".join(tokens)
   text_t.strip()
   text_t = re.sub('\s+', ' ', text_t)
   text_t = re.sub(r'[^\w\s]', ' ', text_t)
   print(text_t)
   tokens = text_t.split(' ')
   tokens = [token for token in tokens if len(token) >= 2]
   return tokens
for text in docs:
   tokened_docs.append(tokenization(text))

tokened_docs
```

```
  Entertainment is giving people what they want     Howard Hawks Entertainment  in its essence  is the art of captivatin
 What is Metaverse  Metaverse is an online  3D universe that combines multiple different virtual spaces  It is just a co
 A Metaverse is a virtual world that combines thoughts from the Internet  augmented reality  virtual reality  artificial
 Covid 19 has been declared a pandemic by the World Health Organization for more than a year  Lockdowns have affected hu
 Ever since the stone age  humans have been evolving in one way or the other  In the recent past  this wave of evolution
[['Entertainment',
  'is',
  'giving',
  'people',
  'what',
  'they',
  'want',
  'Howard',
  'Hawks',
  'Entertainment',
  'in',
  'its',
  'essence',
  'is',
  'the',
  'art',
  'of',
  'captivating',
  'and',
  'delighting',
  'people',
  'whisking',
  'them',
  'away',
  'from',
  'the',
  'mundane',
  'and',
  'into',
  'world',
  'of',
  'imagination',
  'and',
  'emotion',
  'It',
  'comes',
  'in',
  'various',
  'forms',
  'from',
  'the',
  'tales',
  'told',
  'around',
  'ancient',
  'campfires',
  'to',
  'the',
  'dazzling',
  'spectacles',
  'of',
  'modern',
  'cinema',
```

```
import nltk
from nltk.corpus import stopwords

nltk.download('stopwords')
```

```
    [nltk_data] Downloading package stopwords to /root/nltk_data...
    [nltk_data]   Unzipping corpora/stopwords.zip.
    True
```

```
stop_words = set(stopwords.words('english'))
filtered_docs = []
def remove(tokens):
  return [token for token in tokens if token.lower() not in stop_words]
for tokens in tokened_docs:
  filtered_docs.append(remove(tokens))

filtered_docs
```

```
    [['Entertainment',
      'giving',
      'people',
      'want',
      'Howard',
      'Hawks',
      'Entertainment',
      'essence',
      'art',
      'captivating',
      'delighting',
      'people',
      'whisking',
      'away',
      'mundane',
      'world',
      'imagination',
      'emotion',
      'comes',
      'various',
      'forms',
      'tales',
      'told',
      'around',
      'ancient',
      'campfires',
      'dazzling',
      'spectacles',
      'modern',
      'cinema',
      'Traditionally',
      'relied',
      'television',
      'radio',
      'theatres',
      'satiate',
      'thirst',
      'entertainment',
      'However',
      'new',
      'dawn',
      'emerged',
      'advent',
      'Top',
      'OTT',
      'platforms',
      'deliver',
      'wide',
      'array',
      'digital',
      'content',
      'directly',
      'consumers',
      'internet',
      'transforming',
      'way',
      'consume',
      'content',
```

```
import nltk
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer


wnl = WordNetLemmatizer()
```

```
    [nltk_data] Downloading package wordnet to /root/nltk_data...
```

```
from textblob import TextBlob

def pos_tagger(sentence):
    sent = TextBlob(sentence)
    tag_dict = {"J": 'a', "N": 'n', "V": 'v', "R": 'r'}
    words_tags = [(w, tag_dict.get(pos[0], 'n')) for w, pos in sent.tags]
    lemma_list = [wd.lemmatize(tag) for wd, tag in words_tags]
    return lemma_list


def lemma(tokens):
  return [wnl.lemmatize(word.lower()) for word in tokens]

lemma_docs = []
for tokens in filtered_docs:
  lemma_docs.append(lemma(tokens))

lemma_docs
```

```
[['entertainment',
  'giving',
  'people',
  'want',
  'howard',
  'hawk',
  'entertainment',
  'essence',
  'art',
  'captivating',
  'delighting',
  'people',
  'whisking',
  'away',
  'mundane',
  'world',
  'imagination',
  'emotion',
  'come',
  'various',
  'form',
  'tale',
  'told',
  'around',
  'ancient',
  'campfire',
  'dazzling',
  'spectacle',
  'modern',
  'cinema',
  'traditionally',
  'relied',
  'television',
  'radio',
  'theatre',
  'satiate',
  'thirst',
  'entertainment',
  'however',
  'new',
  'dawn',
  'emerged',
  'advent',
  'top',
  'ott',
  'platform',
  'deliver',
  'wide',
  'array',
  'digital',
  'content',
  'directly',
  'consumer',
  'internet',
  'transforming',
  'way',
  'consume',
  'content',
```

```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

```
import nltk
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
True
```

```
df = {}
terms = []
def table(lemma_docs):
  lst = []
  i = 0
  for doc in lemma_docs:
    df.update({f"text{i}":doc})
    for word in doc:
      if word not in terms:
        terms.append(word.lower())
    i = i + 1
  return df

tab = table(lemma_docs)
terms
```

```
['entertainment',
 'giving',
 'people',
 'want',
 'howard',
 'hawk',
 'essence',
 'art',
 'captivating',
 'delighting',
 'whisking',
 'away',
 'mundane',
 'world',
 'imagination',
 'emotion',
 'come',
 'various',
 'form',
 'tale',
 'told',
 'around',
 'ancient',
 'campfire',
 'dazzling',
 'spectacle',
 'modern',
 'cinema',
 'traditionally',
 'relied',
 'television',
 'radio',
 'theatre',
 'satiate',
 'thirst',
 'however',
 'new',
 'dawn',
 'emerged',
 'advent',
 'top',
 'ott',
 'platform',
 'deliver',
 'wide',
 'array',
 'digital',
 'content',
 'directly',
 'consumer',
 'internet',
 'transforming',
 'way',
 'consume',
 'accessible',
 'tap',
 'screen',
 'redefining',
vec_dict = {}
terms.sort()
vec_dict.update({"words" : terms})
for i in tab:
  lst = []
  for j in terms:
    if j in tab[i]:
      lst.append(1)
    else:
      lst.append(0)
  vec_dict.update({i:lst})

vec_dict
```

```
{'words': ['10',
  '19',
  '1991',
  '2000s',
  '2021',
  '2024',
  '2040',
  '21',
  '29th',
  '2d',
  '300',
  '38',
  '39',
  '3d',
  '47',
  '4sight',
  '800',
  'abhijit',
  'ability',
  'able',
  'abrupt',
  'accepted',
  'access',
  'accessibility',
  'accessible',
  'accessing',
  'accident',
  'acclaimed',
  'accomplishing',
  'according',
  'account',
  'acquired',
  'across',
  'act',
  'action',
  'actionable',
  'activation',
  'active',
  'actively',
  'activity',
  'actor',
  'actual',
  'ad',
  'adapt',
  'adapted',
  'add',
  'addictive',
  'adding',
  'addition',
  'additional',
  'additionally',
  'addressed',
  'administration',
  'adopt',
  'adopting',
  'adoption',
  'advance',
  'advanced',
```

```
import pandas as pd
data  = pd.DataFrame(vec_dict)
data.sample(10)
```

	words	text0	text1	text2	text3	text4
470	easy	0	0	0	1	1
1183	problem	0	0	0	1	0
868	korean	0	0	1	0	0
287	communication	0	0	1	1	1

```
invert_idx = {}
for j in range(len(terms)):
  word = terms[j]
  lst = []
  for i in range(len(docs)):
    if vec_dict[f"text{i}"][j] == 1:
       lst.append(f"text{i}")
  invert_idx.update({word:lst})
invert_idx
```

```
{'10': ['text2'],
 '19': ['text3', 'text4'],
 '1991': ['text4'],
 '2000s': ['text0'],
 '2021': ['text2', 'text3', 'text4'],
 '2024': ['text2', 'text3'],
 '2040': ['text0'],
 '21': ['text3'],
 '29th': ['text4'],
 '2d': ['text3'],
 '300': ['text2'],
 '38': ['text4'],
 '39': ['text1', 'text2', 'text3'],
 '3d': ['text1', 'text2', 'text3'],
 '47': ['text0'],
 '4sight': ['text3'],
 '800': ['text2', 'text3'],
 'abhijit': ['text3'],
 'ability': ['text0', 'text2'],
 'able': ['text2', 'text3', 'text4'],
 'abrupt': ['text3'],
 'accepted': ['text3'],
 'access': ['text0', 'text1', 'text3', 'text4'],
 'accessibility': ['text0'],
 'accessible': ['text0', 'text1', 'text4'],
 'accessing': ['text4'],
 'accident': ['text3'],
 'acclaimed': ['text0'],
 'accomplishing': ['text2'],
 'according': ['text2', 'text3', 'text4'],
 'account': ['text4'],
 'acquired': ['text4'],
 'across': ['text0', 'text1', 'text3'],
 'act': ['text0'],
 'action': ['text4'],
 'actionable': ['text3'],
 'activation': ['text4'],
 'active': ['text0', 'text2', 'text3'],
 'actively': ['text4'],
 'activity': ['text1', 'text2'],
 'actor': ['text4'],
 'actual': ['text2', 'text3'],
 'ad': ['text0'],
 'adapt': ['text0'],
 'adapted': ['text4'],
 'add': ['text3'],
 'addictive': ['text4'],
 'adding': ['text4'],
 'addition': ['text3', 'text4'],
 'additional': ['text3'],
 'additionally': ['text0', 'text2'],
 'addressed': ['text3'],
 'administration': ['text2'],
 'adopt': ['text0', 'text4'],
 'adopting': ['text4'],
 'adoption': ['text3'],
 'advance': ['text2'],
 'advanced': ['text4'],
def intersection(lst1, lst2):
  lst3 = [value for value in lst1 if value in lst2]
  return lst3

def Union(lst1, lst2):
  final_list = lst1 + lst2
  return final_list

q = "menu and mere"

q_t = q.split(' ')

lst = []
ops = []
i = 0;
while i < len(q_t):
  lst.append(q_t[i])
  if i+1 < len(q_t):
    ops.append(q_t[i+1])
  i = i + 2

print(lst)
print(ops)

ans = invert_idx[lst[0]]
for i in range(1,len(lst)):
   if ops[i-1] == "or":
     ans = Union(ans,invert_idx[lst[i]])
   if ops[i-1] == "and":
     ans = intersection(ans,invert_idx[lst[i]])

ans
```

```
     ['menu', 'mere']
     ['and']
     ['text0']
```

```
def encode(DocID):
    bytes_list = []
    s = ""
    flag = 0
    while DocID > 0:
        num = DocID % 2
        s = str(num) + s
        DocID = DocID // 2
        if len(s) == 7:
          if flag == 0:
            s = "1" + s
          else:
            s = "0" + s
          bytes_list.insert(0,s)
          s = ""
    if s != "":
      while len(s) < 7:
        s = "0" + s
      if flag == 0:
        s = "1" + s
      else:
        s = "0" + s
      bytes_list.insert(0,s)
    return bytes_list


bits = encode(206)
print(bits)
```

```
['10000001', '11001110']
```

```
from bitarray import bitarray

def gamma_code(n):
  binary_n = format(n, 'b')
  binary_offset = binary_n[1::]
  unary_length = bitarray(True for i in range(len(binary_offset))) + bitarray([False])
  return bitarray(unary_length), bitarray(binary_offset)

code = gamma_code(26)
print(code)
```

```
(bitarray('11110'), bitarray('1010'))
```

```
!pip3 install bitarray
```

```
Collecting bitarray
  Downloading bitarray-2.8.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (286 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 286.5/286.5 kB 5.1 MB/s eta 0:00:00
Installing collected packages: bitarray
Successfully installed bitarray-2.8.2
```

```
texts = [
    "Data science is all about extracting insights from data to make informed decisions. It involves a mix of statistics, pr
    "Natural Language Processing, or NLP, enables computers to understand, interpret, and generate human language. It's usec
    "Machine learning is a subset of AI where computers learn patterns from data to make predictions. It's widely used in re
    "Deep learning is a subfield of machine learning that uses artificial neural networks to model complex patterns. It's th
    "Artificial Intelligence is the broader concept of machines being able to perform tasks that typically require human int
    "Data scientists use tools like Python, R, and data visualization techniques to uncover hidden insights within large dat
    "Supervised learning, unsupervised learning, and reinforcement learning are three main paradigms of machine learning, ea
    "AI systems can be categorized into narrow or weak AI (designed for specific tasks) and general or strong AI (with humar
    "Convolutional Neural Networks (CNNs) are pivotal for image recognition, while Recurrent Neural Networks (RNNs) excel ir
    "NLP applications include sentiment analysis, text summarization, and chatbots that can hold natural conversations with
    "Ensemble learning combines the predictions of multiple machine learning models to improve accuracy, with methods like b
    "Ethical considerations are crucial in AI development, including issues related to bias, privacy, and the responsible us
    "Big data analytics is a significant part of data science, involving the processing and analysis of large and complex da
    "Named Entity Recognition (NER) is a common NLP task that involves identifying and categorizing named entities such as r
    "Generative Adversarial Networks (GANs) are a popular deep learning technique for generating synthetic data and images."
    "Reinforcement learning is a subset of machine learning where agents learn by interacting with their environment and rec
    "Data cleaning, preprocessing, and feature engineering are essential steps in data science to ensure high-quality data f
    "Regression and classification are two fundamental types of supervised machine learning, used for predicting continuous
    "AI has the potential to revolutionize healthcare, with applications in disease diagnosis, drug discovery, and personali
    "Transformers, a deep learning architecture, have revolutionized NLP by enabling models like BERT and GPT-3, which excel
]
```

```
import re

tokened_docs = []
def tokenization(text):
   tokens = text.split(' ')
   text_t = " ".join(tokens)
   text_t.strip()
   text_t = re.sub('\s+', ' ', text_t)
   text_t = re.sub(r'[^\w\s]', ' ', text_t)
   print(text_t)
   tokens = text_t.split(' ')
   tokens = [token for token in tokens if len(token) >= 2]
   return tokens
for text in texts:
   tokened_docs.append(tokenization(text))

tokened_docs
```

```
       'classification',
       'are',
       'two',
       'fundamental',
       'types',
       'of',
       'supervised',
       'machine',
       'learning',
       'used',
       'for',
       'predicting',
       'continuous',
       'and',
       'categorical',
       'outcomes',
       'respectively'],
      ['AI',
       'has',
       'the',
       'potential',
       'to',
       'revolutionize',
       'healthcare',
       'with',
       'applications',
       'in',
       'disease',
       'diagnosis',
       'drug',
       'discovery',
       'and',
       'personalized',
       'treatment',
       'plans'],
      ['Transformers',
       'deep',
       'learning',
       'architecture',
       'have',
       'revolutionized',
       'NLP',
       'by',
       'enabling',
       'models',
       'like',
       'BERT',
       'and',
       'GPT',
       'which',
       'excel',
       'in',
       'wide',
       'range',
       'of',
       'language',
       'understanding',
       'tasks']]
```

```
import nltk
from nltk.corpus import stopwords

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'))
filtered_docs = []
def remove(tokens):
  return [token for token in tokens if token.lower() not in stop_words]
for tokens in tokened_docs:
  filtered_docs.append(remove(tokens))

filtered_docs
```

```
  'punishments'],
 ['Data',
  'cleaning',
  'preprocessing',
  'feature',
  'engineering',
  'essential',
  'steps',
  'data',
  'science',
  'ensure',
  'high',
  'quality',
  'data',
  'analysis'],
 ['Regression',
  'classification',
  'two',
  'fundamental',
  'types',
  'supervised',
  'machine',
  'learning',
  'used',
  'predicting',
  'continuous',
  'categorical',
  'outcomes',
  'respectively'],
 ['AI',
  'potential',
  'revolutionize',
  'healthcare',
  'applications',
  'disease',
  'diagnosis',
  'drug',
  'discovery',
  'personalized',
  'treatment',
  'plans'],
 ['Transformers',
  'deep',
  'learning',
  'architecture',
  'revolutionized',
  'NLP',
  'enabling',
  'models',
  'like',
  'BERT',
  'GPT',
  'excel',
  'wide',
  'range',
  'language',
  'understanding',
  'tasks']]
```

```
import nltk
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer


wnl = WordNetLemmatizer()
```

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

```
from textblob import TextBlob

def pos_tagger(sentence):
    sent = TextBlob(sentence)
    tag_dict = {"J": 'a', "N": 'n', "V": 'v', "R": 'r'}
    words_tags = [(w, tag_dict.get(pos[0], 'n')) for w, pos in sent.tags]
    lemma_list = [wd.lemmatize(tag) for wd, tag in words_tags]
    return lemma_list

def lemma(tokens):
  return [wnl.lemmatize(word.lower()) for word in tokens]

lemma_docs = []
for tokens in filtered_docs:
  lemma_docs.append(lemma(tokens))

lemma_docs
```

```
    [['data',
      'science',
      'extracting',
      'insight',
      'data',
      'make',
      'informed',
      'decision',
      'involves',
      'mix',
      'statistic',
      'programming',
      'domain',
      'expertise'],
     ['natural',
      'language',
      'processing',
      'nlp',
      'enables',
      'computer',
      'understand',
      'interpret',
      'generate',
      'human',
      'language',
      'used',
      'chatbots',
      'translation',
      'sentiment',
      'analysis'],
     ['machine',
      'learning',
      'subset',
      'ai',
      'computer',
      'learn',
      'pattern',
      'data',
      'make',
      'prediction',
      'widely',
      'used',
      'recommendation',
      'system',
      'fraud',
      'detection'],
     ['deep',
      'learning',
      'subfield',
      'machine',
      'learning',
      'us',
      'artificial',
      'neural',
      'network',
      'model',
      'complex',
      'pattern',
unique_terms = []
for doc in lemma_docs:
  for word in doc:
    if word in unique_terms:
      continue
    unique_terms.append(word)

unique_terms.sort()
unique_terms
```

```
    ['able',
     'accuracy',
     'adversarial',
     'agent',
     'ai',
     'analysis',
     'analytics',
     'application',
     'architecture',
     'artificial',
     'bagging',
     'behind',
     'bert',
     'bias',
     'big',
     'boosting',
     'broader',
     'categorical',
     'categorized',
     'categorizing',
     'chatbots',
     'classification',
     'cleaning',
     'cnns',
     'combine',
     'common',
     'complex',
     'computer',
     'concept',
     'consideration',
     'continuous',
     'conversation',
     'convolutional',
     'crucial',
     'data',
     'datasets',
     'date',
     'decision',
     'deep',
     'designed',
     'detection',
     'development',
     'diagnosis',
     'discovery',
     'disease',
     'domain',
     'drug',
     'enables',
     'enabling',
     'engineering',
     'ensemble',
     'ensure',
     'entity',
     'environment',
     'essential',
     'ethical',
     'excel',
     'expertise',
```

```
dic = {}
for ind,text in enumerate(texts):
  dummy = []
  for word in unique_terms:
    if word in text.lower():
      dummy.append(1)
    else:
      dummy.append(0)
  dic.update({f"Doc{ind+1}": dummy})

dic
```

```
    {'Doc1': [0,
      0,
      0,
      0,
      1,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      1,
      0,
      0,
      1,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      1,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      0,
      1,

import pandas as pd
df = pd.DataFrame(dic).T
df.head(20)
```

	0	1	2	3	4	5	6	7	8	9	...	178	179	180	181	182	183	184	185	186	187
Doc1	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
Doc2	1	0	0	0	0	1	0	0	0	0	...	1	1	1	0	0	0	0	0	0	0
Doc3	0	0	0	0	1	0	0	0	0	0	...	1	1	1	0	0	0	0	1	1	0

```
query = "deep learning"
q1 = "deep"
q2 = "learning"
retrieved_docs = []
for ind, doc in enumerate(lemma_docs):
   if q1 in doc and q2 in doc:
      retrieved_docs.append(f"Doc{ind+1}")

retrieved_docs
```

```
    ['Doc4', 'Doc15', 'Doc20']
```

```
relevant_docs = ["Doc4","Doc9","Doc11","Doc15"]
relevant_retrieved = [doc for doc in relevant_docs if doc in retrieved_docs]
relevant_not_retrieved = [doc for doc in relevant_docs if doc not in retrieved_docs]
irrelevant_retreived = [doc for doc in retrieved_docs if doc not in relevant_docs]

recall = len(relevant_retrieved)/len(relevant_docs)
precision = len(relevant_retrieved)/len(retrieved_docs)

print(recall)
print(precision)

f_measure = (2*(precision*recall))/(precision+recall)

print(f_measure)
```

```
    0.5
    0.6666666666666666
    0.5714285714285715
```

```python
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left_half = arr[:mid]
        right_half = arr[mid:]
        merge_sort(left_half)
        merge_sort(right_half)
        i = j = k = 0
        while i < len(left_half) and j < len(right_half):
            if left_half[i] < right_half[j]:
                arr[k] = left_half[i]
                i += 1
            else:
                arr[k] = right_half[j]
                j += 1
            k += 1
        while i < len(left_half):
            arr[k] = left_half[i]
            i += 1
            k += 1
        while j < len(right_half):
            arr[k] = right_half[j]
            j += 1
            k += 1

def merge_sorted_blocks(posting_list, k):
    sorted_blocks = []
    for i in range(0, len(posting_list), k):
        block = posting_list[i:i + k]
        print("The Block :",block)
        merge_sort(block)
        print("Sorted Block :",block)
        sorted_blocks.append(block)
        print()

    merged_list = []

    while len(sorted_blocks) > 1:
        block1 = sorted_blocks.pop(0)
        block2 = sorted_blocks.pop(0)

        merged_block = []
        i = j = 0

        while i < len(block1) and j < len(block2):
            if block1[i] < block2[j]:
                merged_block.append(block1[i])
                i += 1
            else:
                merged_block.append(block2[j])
                j += 1

        merged_block.extend(block1[i:])
        merged_block.extend(block2[j:])
        sorted_blocks.append(merged_block)

    return sorted_blocks[0]

posting_list = [18, 2, 89, 45, 61, 23, 7, 56, 43, 9]
k = 3
print("Posting list :",posting_list)
print()
sorted_posting_list = merge_sorted_blocks(posting_list, k)
print(sorted_posting_list)
```

```
Posting list : [18, 2, 89, 45, 61, 23, 7, 56, 43, 9]

The Block : [18, 2, 89]
Sorted Block : [2, 18, 89]

The Block : [45, 61, 23]
Sorted Block : [23, 45, 61]

The Block : [7, 56, 43]
Sorted Block : [7, 43, 56]

The Block : [9]
Sorted Block : [9]

[2, 7, 9, 18, 23, 43, 45, 56, 61, 89]
```

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


from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer


documents = [
    "A blcokchain is a shared distributed database or ledger between computer network nodes. Similar to a database, a blockc
    "In cryptocurrency systems like Bitcoin, where they maintain a secure and decetralused record of transactions, blockchai
    "The peculiarity of the blockchain is that it fosters confidence without the need for a reliable third party by guarante
    "A regular database and a blockchain both use different organisational methods for their data. Blocks, which are collect
    "A block is closed and connected to the previous full block when its storage capacity is reached, creating the data chai
    "The goal of blockchain is to make it possible to store, share, and maintain the integrity of digital information.",
    "In this respect a blockchain acts as the basis for immutable ledgers, or records of transactions that cannot be alterec
]
print(documents[1])
```

```
    In cryptocurrency systems like Bitcoin, where they maintain a secure and decetralused record of transactions, blockchain
```

```
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

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

```
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))


def preprocess(document):
    words = word_tokenize(document.lower())
    words = [word for word in words if word.isalpha()]
    words = [word for word in words if word not in stop_words]
    words = [lemmatizer.lemmatize(word) for word in words]
    return ' '.join(words)


preprocessed_documents = [preprocess(doc) for doc in documents]
preprocessed_documents
```

```
    ['blcokchain shared distributed database ledger computer network node similar database blockchain store data
    electronically digital form',
     'cryptocurrency system like bitcoin maintain secure decetralused record transaction blockchains well known playing
    crucial role',
     'peculiarity blockchain foster confidence without need reliable third party guaranteeing security correctness data
    record',
     'regular database blockchain use different organisational method data block collection data store set information
    blockchain gather information',
     'block closed connected previous full block storage capacity reached creating data chain known blockchain',
     'goal blockchain make possible store share maintain integrity digital information',
     'respect blockchain act basis immutable ledger record transaction altered nullified erased']
```

```
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(preprocessed_documents)


print("Vectored Matrix: ")
print(X.toarray())
X = X.toarray()
```

```
    Vectored Matrix:
    [[0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 2 0 0 1 1 1 0 1 0 0 0 0 0 0 0 0
      0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0]
     [0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
      1 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 1 0 1 0 1 0]
     [0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0
      0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1]
     [0 0 0 0 0 1 2 0 0 0 0 1 0 0 0 0 0 0 0 2 1 0 1 0 0 0 0 0 0 0 1 0 0 0 2 0
      0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0]
     [0 0 0 0 0 2 1 0 1 1 1 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
      1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 1
      0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0]
```

```
 [1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0
  0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0]]
```

```
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd
import numpy as np


def cosine_similarity(vector1, vector2):
    dot_product = np.dot(vector1, vector2)
    norm_vector1 = np.linalg.norm(vector1)
    norm_vector2 = np.linalg.norm(vector2)
    similarity = dot_product / (norm_vector1 * norm_vector2)
    return similarity


document_labels = ["D1", "D2", "D3", "D4", "D5", "D6", "D7"]

num_documents = len(X)
cosine_sim_matrix = np.zeros((num_documents, num_documents))
for i in range(num_documents):
    for j in range(num_documents):
        cosine_sim_matrix[i][j] = cosine_similarity(X[i], X[j])

print("Cosine Similarity Matrix:-\n")
print("       ", end="")
for label in document_labels:
    print(f"{label:8}", end="")
print()
for i in range(num_documents):
    print(f"{document_labels[i]:5}", end="")
    for j in range(num_documents):
        print(f"{cosine_sim_matrix[i][j]:8.4f}", end="")
    print()
```

```
Cosine Similarity Matrix:-

         D1      D2      D3      D4      D5      D6      D7
D1     1.0000  0.0000  0.1260  0.3440  0.1179  0.2236  0.1421
D2     0.0000  1.0000  0.0690  0.0000  0.0645  0.0816  0.1557
D3     0.1260  0.0690  1.0000  0.2229  0.1336  0.0845  0.1612
D4     0.3440  0.0000  0.2229  1.0000  0.3128  0.3297  0.1257
D5     0.1179  0.0645  0.1336  0.3128  1.0000  0.0791  0.0754
D6     0.2236  0.0816  0.0845  0.3297  0.0791  1.0000  0.0953
D7     0.1421  0.1557  0.1612  0.1257  0.0754  0.0953  1.0000
```

```
def jaccard_similarity(set1, set2):
    intersection = len(set1.intersection(set2))
    union = len(set1.union(set2))
    if union == 0:
        return 0
    return intersection / union


def cosine_sim(doc1, doc2):
  words = set (doc1 + doc2)
  c = 0
  for w in words:
    if w in doc1 and w in doc2:
      c+=1

  n1 = len(set(doc1))
  n2 = len(set(doc2))

  sim = c/(n1*n2)

  return round(sim,4)


document_sets = [
    set([0,0,0,0,1,0,1,0,0,0,0,0,1,0,0,0,0,0,0,1,2,0,0,1,1,1,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
    set([0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,1,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,1,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,
    set([0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,1,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,1,0,1,0,
    set([0,0,0,0,0,1,2,0,0,0,0,1,0,0,0,0,0,0,0,2,1,0,1,0,0,0,0,0,0,0,1,0,0,0,2,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,
    set([0,0,0,0,0,2,1,0,1,1,1,0,0,0,1,0,1,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,
    set([0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,1,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,
    set([1,1,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,1,
]
```

```
num_documents = len(document_sets)
jaccard_sim_matrix = np.zeros((num_documents, num_documents))
for i in range(num_documents):
    for j in range(num_documents):
        jaccard_sim_matrix[i][j] = jaccard_similarity(document_sets[i], document_sets[j])


print("Jaccard Similarity Matrix:-\n")
print("       ", end="")
for label in document_labels:
    print(f"{label:12}", end="")
print()
for i in range(num_documents):
    print(f"{document_labels[i]:8}", end="")
    for j in range(num_documents):
        print(f"{jaccard_sim_matrix[i][j]:.4f}      ", end="")
    print()
```

```
Jaccard Similarity Matrix:-

        D1          D2          D3          D4          D5          D6          D7
D1      1.0000      0.6667      0.6667      1.0000      1.0000      0.6667      0.6667
D2      0.6667      1.0000      1.0000      0.6667      0.6667      1.0000      1.0000
D3      0.6667      1.0000      1.0000      0.6667      0.6667      1.0000      1.0000
D4      1.0000      0.6667      0.6667      1.0000      1.0000      0.6667      0.6667
D5      1.0000      0.6667      0.6667      1.0000      1.0000      0.6667      0.6667
D6      0.6667      1.0000      1.0000      0.6667      0.6667      1.0000      1.0000
D7      0.6667      1.0000      1.0000      0.6667      0.6667      1.0000      1.0000
```

```
def euclidean_distance(vector1, vector2):
    squared_diff = [(x - y)**2 for x, y in zip(vector1, vector2)]
    distance = np.sqrt(sum(squared_diff))
    return distance


num_documents = X.shape[0]
euclidean_distances = np.zeros((num_documents, num_documents))
for i in range(num_documents):
    for j in range(num_documents):
        euclidean_distances[i][j] = euclidean_distance(X[i], X[j])


print("Euclidean Distance Matrix:")
print("       ", end="")
for label in document_labels:
    print(f"{label:8}", end="")
print()
for i in range(num_documents):
    print(f"{document_labels[i]:5}", end="")
    for j in range(num_documents):
        print(f"{euclidean_distances[i][j]:8.2f}", end="")
    print()
```

```
Euclidean Distance Matrix:
        D1      D2      D3      D4      D5      D6      D7
D1       0.00    5.74    5.29    5.20    5.48    4.69    5.00
D2       5.74    0.00    5.20    6.16    5.39    4.80    4.69
D3       5.29    5.20    0.00    5.39    5.10    4.69    4.58
D4       5.20    6.16    5.39    0.00    5.20    4.80    5.48
D5       5.48    5.39    5.10    5.20    0.00    4.90    5.00
D6       4.69    4.80    4.69    4.80    4.90    0.00    4.36
D7       5.00    4.69    4.58    5.48    5.00    4.36    0.00
```

```
inv_index = { }
words = set("A blcokchain is a shared distributed database or ledger between computer network nodes. Similar to a database,

for w in words:
  inv_index[w] = []
  for d in range(len(documents)):
    if w in documents[d]:
      inv_index[w] += [d+1]
```

```
high_list = []
low_list = []

query = "blockchain information"
terms = query.split()
term1 = terms[0]
term2 = terms[1]

for i, document in enumerate(documents):
    if term1 in document.lower() and term2 in document.lower() :
        high_list.append((i + 1, document))
    else:
        low_list.append((i + 1, document))

print("High List (Documents containing 'blockchain'):")
for doc_index, doc in high_list:
    print(f"Document {doc_index}: {doc}")

print("\nLow List (Documents not containing 'blockchain'):")
for doc_index, doc in low_list:
    print(f"Document {doc_index}: {doc}")
```

```
High List (Documents containing 'blockchain'):
Document 4: A regular database and a blockchain both use different organisational methods for their data. Blocks, which
Document 6: The goal of blockchain is to make it possible to store, share, and maintain the integrity of digital informa

Low List (Documents not containing 'blockchain'):
Document 1: A blcokchain is a shared distributed database or ledger between computer network nodes. Similar to a databas
Document 2: In cryptocurrency systems like Bitcoin, where they maintain a secure and decetralused record of transactions
Document 3: The peculiarity of the blockchain is that it fosters confidence without the need for a reliable third party
Document 5: A block is closed and connected to the previous full block when its storage capacity is reached, creating th
Document 7: In this respect a blockchain acts as the basis for immutable ledgers, or records of transactions that cannot
```

```
high_list
```

```
[(4,
  'A regular database and a blockchain both use different organisational methods for their data. Blocks, which are
collections of data that store sets of information, are how a blockchain gathers information.'),
 (6,
  'The goal of blockchain is to make it possible to store, share, and maintain the integrity of digital information.')]
```

```
low_list
```

```
[(1,
  'A blcokchain is a shared distributed database or ledger between computer network nodes. Similar to a database, a
blockchain stores data electronically in digital form.'),
 (2,
  'In cryptocurrency systems like Bitcoin, where they maintain a secure and decetralused record of transactions,
blockchains are well known for playing a crucial role.'),
 (3,
  'The peculiarity of the blockchain is that it fosters confidence without the need for a reliable third party by
guaranteeing the security and correctness of a data record.'),
 (5,
  'A block is closed and connected to the previous full block when its storage capacity is reached, creating the data
chain known as the blockchain.'),
 (7,
  'In this respect a blockchain acts as the basis for immutable ledgers, or records of transactions that cannot be
altered, nullified, or erased.')]
```

```
n = 2
print("Number of docs required: ", n)
l = []
print("Relevant docs: ")
for i in high_list:
  print(i)
  n-=1
  if(n==0): break

if(n>0):
  for i in low_list:
    print(i)
    n-=1
    if(n==0): break
```

```
Number of docs required:  2
Relevant docs:
(4, 'A regular database and a blockchain both use different organisational methods for their data. Blocks, which are col
(6, 'The goal of blockchain is to make it possible to store, share, and maintain the integrity of digital information.')
```

```
from collections import Counter
```

```
frequency1=[Counter(preprocessed_documents[0].split())]
print(frequency1)
frequency2=[Counter(preprocessed_documents[1].split())]
print(frequency2)
frequency3=[Counter(preprocessed_documents[2].split())]
print(frequency3)
frequency4=[Counter(preprocessed_documents[3].split())]
print(frequency4)
frequency5=[Counter(preprocessed_documents[4].split())]
print(frequency5)
frequency6=[Counter(preprocessed_documents[5].split())]
print(frequency6)
frequency7=[Counter(preprocessed_documents[6].split())]
print(frequency7)
```

```
    [Counter({'database': 2, 'blcokchain': 1, 'shared': 1, 'distributed': 1, 'ledger': 1, 'computer': 1, 'network': 1, 'node
    [Counter({'cryptocurrency': 1, 'system': 1, 'like': 1, 'bitcoin': 1, 'maintain': 1, 'secure': 1, 'decetralused': 1, 'rec
    [Counter({'peculiarity': 1, 'blockchain': 1, 'foster': 1, 'confidence': 1, 'without': 1, 'need': 1, 'reliable': 1, 'thir
    [Counter({'blockchain': 2, 'data': 2, 'information': 2, 'regular': 1, 'database': 1, 'use': 1, 'different': 1, 'organisa
    [Counter({'block': 2, 'closed': 1, 'connected': 1, 'previous': 1, 'full': 1, 'storage': 1, 'capacity': 1, 'reached': 1,
    [Counter({'goal': 1, 'blockchain': 1, 'make': 1, 'possible': 1, 'store': 1, 'share': 1, 'maintain': 1, 'integrity': 1, '
    [Counter({'respect': 1, 'blockchain': 1, 'act': 1, 'basis': 1, 'immutable': 1, 'ledger': 1, 'record': 1, 'transaction':
```

```
zipfian1=[]
for c in frequency1:
  freq={word:1/(rank+1)for rank,(word,_)in enumerate(c.most_common())}
  zipfian1.append(freq)

print(zipfian1)
zipfian2=[]
for c in frequency2:
  freq={word:1/(rank+1)for rank,(word,_)in enumerate(c.most_common())}
  zipfian2.append(freq)

print(zipfian2)
zipfian3=[]
for c in frequency3:
  freq={word:1/(rank+1)for rank,(word,_)in enumerate(c.most_common())}
  zipfian3.append(freq)

print(zipfian3)
zipfian4=[]
for c in frequency4:
  freq={word:1/(rank+1)for rank,(word,_)in enumerate(c.most_common())}
  zipfian4.append(freq)

print(zipfian4)
zipfian5=[]
for c in frequency5:
  freq={word:1/(rank+1)for rank,(word,_)in enumerate(c.most_common())}
  zipfian5.append(freq)

print(zipfian5)
zipfian6=[]
for c in frequency6:
  freq={word:1/(rank+1)for rank,(word,_)in enumerate(c.most_common())}
  zipfian6.append(freq)

print(zipfian6)

zipfian7=[]
for c in frequency7:
  freq={word:1/(rank+1)for rank,(word,_)in enumerate(c.most_common())}
  zipfian7.append(freq)

print(zipfian7)
```

```
    [{'database': 1.0, 'blcokchain': 0.5, 'shared': 0.3333333333333333, 'distributed': 0.25, 'ledger': 0.2, 'computer': 0.16
    [{'cryptocurrency': 1.0, 'system': 0.5, 'like': 0.3333333333333333, 'bitcoin': 0.25, 'maintain': 0.2, 'secure': 0.166666
    [{'peculiarity': 1.0, 'blockchain': 0.5, 'foster': 0.3333333333333333, 'confidence': 0.25, 'without': 0.2, 'need': 0.166
    [{'blockchain': 1.0, 'data': 0.5, 'information': 0.3333333333333333, 'regular': 0.25, 'database': 0.2, 'use': 0.16666666
    [{'block': 1.0, 'closed': 0.5, 'connected': 0.3333333333333333, 'previous': 0.25, 'full': 0.2, 'storage': 0.166666666666
    [{'goal': 1.0, 'blockchain': 0.5, 'make': 0.3333333333333333, 'possible': 0.25, 'store': 0.2, 'share': 0.166666666666666
    [{'respect': 1.0, 'blockchain': 0.5, 'act': 0.3333333333333333, 'basis': 0.25, 'immutable': 0.2, 'ledger': 0.16666666666
```

```
diffs1=[]
for actual,freq in zip(frequency1,zipfian1):
  diff={word: (actual[word]-freq.get(word,0))*100 for word in actual}
  diffs1.append(diff)
print(diffs1)

diffs2=[]
for actual,freq in zip(frequency2,zipfian2):
```

```
for actual,freq in zip(frequency2,zipfian2):
  diff={word: (actual[word]-freq.get(word,0))*100 for word in actual}
  diffs2.append(diff)
print(diffs2)

diffs3=[]
for actual,freq in zip(frequency3,zipfian3):
  diff={word: (actual[word]-freq.get(word,0))*100 for word in actual}
  diffs3.append(diff)
print(diffs3)

diffs4=[]
for actual,freq in zip(frequency4,zipfian4):
  diff={word: (actual[word]-freq.get(word,0))*100 for word in actual}
  diffs4.append(diff)
print(diffs4)

diffs5=[]
for actual,freq in zip(frequency5,zipfian5):
  diff={word: (actual[word]-freq.get(word,0))*100 for word in actual}
  diffs5.append(diff)
print(diffs5)

diffs6=[]
for actual,freq in zip(frequency6,zipfian6):
  diff={word: (actual[word]-freq.get(word,0))*100 for word in actual}
  diffs6.append(diff)
print(diffs6)

diffs7=[]
for actual,freq in zip(frequency7,zipfian7):
  diff={word: (actual[word]-freq.get(word,0))*100 for word in actual}
  diffs7.append(diff)
print(diffs7)
```

```
    [{'blcokchain': 50.0, 'shared': 66.66666666666667, 'distributed': 75.0, 'database': 100.0, 'ledger': 80.0, 'computer': 8
    [{'cryptocurrency': 0.0, 'system': 50.0, 'like': 66.66666666666667, 'bitcoin': 75.0, 'maintain': 80.0, 'secure': 83.3333
    [{'peculiarity': 0.0, 'blockchain': 50.0, 'foster': 66.66666666666667, 'confidence': 75.0, 'without': 80.0, 'need': 83.3
    [{'regular': 75.0, 'database': 80.0, 'blockchain': 100.0, 'use': 83.33333333333334, 'different': 85.71428571428572, 'org
    [{'block': 100.0, 'closed': 50.0, 'connected': 66.66666666666667, 'previous': 75.0, 'full': 80.0, 'storage': 83.33333333
    [{'goal': 0.0, 'blockchain': 50.0, 'make': 66.66666666666667, 'possible': 75.0, 'store': 80.0, 'share': 83.3333333333333
    [{'respect': 0.0, 'blockchain': 50.0, 'act': 66.66666666666667, 'basis': 75.0, 'immutable': 80.0, 'ledger': 83.333333333
```