Task 1

In [161]:
```
import pickle
with open('cleaned_strings', 'rb') as f:
    data = pickle.load(f)
```

In [162]:
```
from tqdm import tqdm # tqdm is a library that helps us to visualize the runtime
#https://tqdm.github.io/
from math import log
from collections import OrderedDict, Counter
from operator import itemgetter
from scipy.sparse import csr_matrix
dic = {}
idf = {}
idfvalue = []
# it accepts only list of sentances
def fit(dataset):
    unique_words = set() # at first we will initialize an empty set
    # check if its list type or not
    if isinstance(dataset, (list,)):
        for row in dataset: # for each review in the dataset
            for word in row.split(" "): # for each word in the review. #split met
                if len(word) < 2:
                    continue
                unique_words.add(word)
        unique_words = sorted(list(unique_words))
        vocab = {j:i for i,j in enumerate(unique_words)}
        #this for is use to find the count of each word in corpus
        for i in vocab:
            count = 0
            for j in range(0,len(dataset)):
                x = dataset[j].find(i)
                if x > -1:
                    count+=1
            dic[i] = count
        # we order the dictionary in asc
        if len(idfvalue) == 0:
            for i in range(len(unique_words)):
                idfvalue.append(idf_values(unique_words[i],data))
        for i in range(len(uniqueword)):
            idf[unique_words[i]] = idfvalue[i]
        return dic,vocab,idfvalue, idf
    else:
        print("you need to pass list of sentance")
```

In [163]:
```
fit(data)[3] # idf value of all word
```

Out[163]:
```
{'aailiyah': 6.61472560020376,
 'abandoned': 6.61472560020376,
 'ability': 5.516113311535651,
 'abroad': 6.61472560020376,
 'absolutely': 4.417501022867541,
 'abstruse': 6.61472560020376,
 'abysmal': 5.921578419643816,
 'academy': 6.61472560020376,
 'accents': 6.61472560020376,
 'accessible': 6.61472560020376,
 'acclaimed': 6.61472560020376,
 'accolades': 6.61472560020376,
 'accurate': 6.61472560020376,
 'accurately': 6.61472560020376,
 'accused': 5.921578419643816,
 'achievement': 5.921578419643816,
 'achille': 6.61472560020376,
 'ackerman': 6.61472560020376,
 'act': 5.921578419643816,
```

In [164]:
```
#reference from https://www.geeksforgeeks.org/tf-idf-model-for-page-ranking/
def idf_values(term, allDocs):
    num_docs_with_given_term = 0
    """
    This function calculate the idf value of all the term in corpus.
    """
    # Iterate through all the documents
    for i in range(len(allDocs)):
        if term.lower() in allDocs[i].lower().split():
            num_docs_with_given_term += 1

    if num_docs_with_given_term > 0:
        # Total number of documents
        total_num_docs = len(allDocs)

        # Calculating the IDF
        idf_val = log(float(total_num_docs) / num_docs_with_given_term)
        return idf_val
    else:
        return 0
```

In [165]:
```
def transform(dataset,vocab, wrd_in_doc):
    rows = []
    columns = []
    values = []
    if isinstance(dataset, (list,)):
        for idx, row in enumerate(tqdm(dataset)): # for each document in the data
            # it will return a dict type object where key is the word and values
            word_freq = dict(Counter(row.split()))
            # for every unique word in the document
            for word, freq in word_freq.items():  # for each unique word in the r
                if len(word) < 2:
                    continue
                # we will check if its there in the vocabulary that we build in f
                # dict.get() function will return the values, if the key doesn't 
                col_index = vocab.get(word, -1) # retreving the dimension number 
                # if the word exists
                if col_index !=-1:
                    # we are storing the index of the document
                    rows.append(idx)
                    # we are storing the dimensions of the word
                    columns.append(col_index)
                    # applying the formula of tf-idf in value
                    values.append(freq/len(row.split())*(log(len(dataset)/wrd_in_
        return csr_matrix((values, (rows,columns)), shape=(len(dataset),len(vocab
    else:
        print("you need to pass list of strings")
```

In [167]:
```
uniqueword = fit(data)[1]
wrd_in_doc = fit(data)[0]
print(transform(data, uniqueword, wrd_in_doc)[0].toarray())
```

```
100%|██████████████████████████████████████████████████████████████████
████| 746/746 [00:00<00:00, 9569.45it/s]

[[0. 0. 0. ... 0. 0. 0.]]
```

In [168]:
```
print(transform(data, uniqueword, wrd_in_doc)[0].get_shape())
```

```
100%|██████████████████████████████████████████████████████████████████
████| 746/746 [00:00<00:00, 9215.10it/s]

(1, 2886)
```

task 2

In [169]:
```
import pickle
with open('cleaned_strings', 'rb') as f:
    data = pickle.load(f)
```

In [170]:
```
from tqdm import tqdm # tqdm is a library that helps us to visualize the runtime
#https://tqdm.github.io/
from collections import OrderedDict, Counter
from operator import itemgetter
from scipy.sparse import csr_matrix
dic = {}
idf = {}
idf__value = []
wrds = []
# it accepts only list of sentances
def fit(dataset):
    unique_words = set() # at first we will initialize an empty set
    # check if its list type or not
    if isinstance(dataset, (list,)):
        for row in dataset: # for each review in the dataset
            for word in row.split(" "): # for each word in the review. #split met
                if len(word) < 2:
                    continue
                unique_words.add(word)
        unique_words = sorted(list(unique_words))
        vocab = {j:i for i,j in enumerate(unique_words)}
        #this for is use to find the count of each word in corpus
        for i in vocab:
            count = 0
            for j in range(0,len(dataset)):
                x = dataset[j].find(i)
                if x > -1:
                    count+=1
            dic[i] = count
        # we order the dictionary in asc
        ordered = OrderedDict(sorted(dic.items(), key=itemgetter(1), reverse=Fals
        if len(wrds) == 0:
            for i in range(100):
                wrds.append(list(ordered.keys())[i])
        # to get top hundred idf score we take top hundred dict key and value
        hundred = dict(list(ordered.items())[:100])
        if len(idf__value) == 0:
            for i in range(len(wrds)):
                idf__value.append(idf_values(wrds[i],data))
        top_hund= { j: i for i, j in enumerate(hundred.keys())}
        for i in range(len(idf__value)):
            idf[wrds[i]] = idf__value[i]
        return top_hund,hundred,idf__value, idf
    else:
        print("you need to pass list of sentance")
```

```
In [171]: fit(data)[3]# idf value of top hundred word
```

```
Out[171]: {'aailiyah': 6.61472560020376,
           'abandoned': 6.61472560020376,
           'abroad': 6.61472560020376,
           'abstruse': 6.61472560020376,
           'academy': 6.61472560020376,
           'accents': 6.61472560020376,
           'accessible': 6.61472560020376,
           'acclaimed': 6.61472560020376,
           'accolades': 6.61472560020376,
           'accurately': 6.61472560020376,
           'achille': 6.61472560020376,
           'ackerman': 6.61472560020376,
           'adams': 6.61472560020376,
           'added': 6.61472560020376,
           'admins': 6.61472560020376,
           'admiration': 6.61472560020376,
           'admitted': 6.61472560020376,
           'adrift': 6.61472560020376,
           'adventure': 6.61472560020376,
           'aesthetically': 6.61472560020376,
           'affected': 6.61472560020376,
           'affleck': 6.61472560020376,
           'afternoon': 6.61472560020376,
           'agreed': 6.61472560020376,
           'aimless': 6.61472560020376,
           'aired': 6.61472560020376,
           'akasha': 6.61472560020376,
           'alert': 6.61472560020376,
           'alike': 6.61472560020376,
           'allison': 6.61472560020376,
           'allowing': 6.61472560020376,
           'alongside': 6.61472560020376,
           'amateurish': 6.61472560020376,
           'amazed': 6.61472560020376,
           'amazingly': 6.61472560020376,
           'amusing': 6.61472560020376,
           'amust': 6.61472560020376,
           'anatomist': 6.61472560020376,
           'angela': 6.61472560020376,
           'angelina': 6.61472560020376,
           'angry': 6.61472560020376,
           'anguish': 6.61472560020376,
           'angus': 6.61472560020376,
           'animals': 6.61472560020376,
           'animated': 6.61472560020376,
           'anita': 6.61472560020376,
           'anniversary': 6.61472560020376,
           'anthony': 6.61472560020376,
           'antithesis': 6.61472560020376,
           'anyway': 6.61472560020376,
           'apart': 6.61472560020376,
           'appears': 6.61472560020376,
           'applauded': 6.61472560020376,
           'applause': 6.61472560020376,
           'argued': 6.61472560020376,
           'armageddon': 6.61472560020376,
           'armand': 6.61472560020376,
           'array': 6.61472560020376,
           'articulated': 6.61472560020376,
           'artiness': 6.61472560020376,
           'artistic': 6.61472560020376,
           'artless': 6.61472560020376,
           'aspects': 6.61472560020376,
           'assante': 6.61472560020376,
           'assaulted': 6.61472560020376,
           'assistant': 6.61472560020376,
           'astonishingly': 6.61472560020376,
           'astronaut': 6.61472560020376,
           'atmosphere': 6.61472560020376,
           'atrocious': 6.61472560020376,
           'atrocity': 6.61472560020376,
           'attempted': 6.61472560020376,
           'attempting': 6.61472560020376,
           'attractive': 6.61472560020376,
           'audio': 6.61472560020376,
           'aurv': 6.61472560020376,
           'austen': 6.61472560020376,
           'austere': 6.61472560020376,
           'author': 6.61472560020376,
           'aversion': 6.61472560020376,
           'avoided': 6.61472560020376,
           'awarded': 6.61472560020376,
           'awards': 6.61472560020376,
           'awkwardly': 6.61472560020376,
           'baaaaaad': 6.61472560020376,
           'babbling': 6.61472560020376,
           'babie': 6.61472560020376,
           'babysitting': 6.61472560020376,
           'backdrop': 6.61472560020376,
           'backed': 6.61472560020376,
           'bailey': 6.61472560020376,
           'bakery': 6.61472560020376,
           'ballet': 6.61472560020376,
           'balls': 6.61472560020376,
           'barcelona': 6.61472560020376,
           'barking': 6.61472560020376,
           'barney': 6.61472560020376,
           'barren': 6.61472560020376,
           'based': 6.61472560020376,
           'bates': 6.61472560020376}
```

In [172]:
```
# note that we are we need to send the preprocessing text here, we have not inlcu
from math import log
def transform(dataset,vocab, wrd_in_doc):
    rows = []
    columns = []
    values = []
    if isinstance(dataset, (list,)):
        for idx, row in enumerate(tqdm(dataset)): # for each document in the data
            # it will return a dict type object where key is the word and values
            word_freq = dict(Counter(row.split()))
            # for every unique word in the document
            for word, freq in word_freq.items():  # for each unique word in the r
                if len(word) < 2:
                    continue
                # we will check if its there in the vocabulary that we build in f
                # dict.get() function will return the values, if the key doesn't
                col_index = vocab.get(word, -1) # retreving the dimension number
                # if the word exists
                if col_index !=-1:
                    # we are storing the index of the document
                    rows.append(idx)
                    # we are storing the dimensions of the word
                    columns.append(col_index)
                    # applying the formula of tf-idf in value
                    values.append(freq/len(row.split())*(log(len(dataset)/wrd_in_
        return csr_matrix((values, (rows,columns)), shape=(len(dataset),len(vocab
    else:
        print("you need to pass list of strings")
```

In [174]:
```
top_hund = fit(data)[0] # top hundred key and value with value as their index
wrd_in_doc = fit(data)[1] # top hundred key and value and value as their num of c
print(transform(data, top_hund, wrd_in_doc)[0].toarray())
```

```
100%|████████████████████████████████████████████████████████████████████
███| 746/746 [00:00<00:00, 46652.71it/s]

[[0.         0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.8268407  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.         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.         0.
  0.         0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.         0.
  0.         0.        ]]
```

Here we can see 25 place as some value because in first sentence their is a word which is from top hundred word.

```
In [175]: print(transform(data, top_hund, wrd_in_doc)[0].get_shape())
```

```
100%|█████████████████████████████████████████████████████████████████████
███| 746/746 [00:00<00:00, 49762.25it/s]

(1, 100)
```

```
In [ ]:
```

```
In [ ]:
```