In [1]:

```python
import pandas as pd
import numpy as np
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
import os
import string
import copy
import pickle
```

In [22]:

```python
title = "comp.graphics"
os.chdir("D:/mini_newsgroups")
paths = []
for (dirpath, dirnames, filenames) in os.walk(str(os.getcwd())+'/'+title+'/'):
    for i in filenames:
        paths.append(str(dirpath)+str("\\")+i)

paths[0]
```

Out[22]:

```
'D:\\mini_newsgroups/comp.graphics/\\37916'
```

In [23]:

```python
len(paths)
```

Out[23]:

```
100
```

:[24] In

```python
#Removing stop words
def remove_stop_words(data):
    stop_words = stopwords.words('english')
    words = word_tokenize(str(data))
    new_text = ""
    for w in words:
        if w not in stop_words:
            new_text = new_text + " " + w
    return np.char.strip(new_text)

#Removing punctuation
def remove_punctuation(data):
    symbols = "!\"#$%&()*+-./:;<=>?@[\]^_`{|}~\n"
    for i in range(len(symbols)):
        data = np.char.replace(data, symbols[i], ' ')
        data = np.char.replace(data, "  ", " ")
    data = np.char.replace(data, ',', '')
    return data

#Convert to lowercase
def convert_lower_case(data):
    return np.char.lower(data)

#Stemming
def stemming(data):
    stemmer= PorterStemmer()

    tokens = word_tokenize(str(data))
    new_text = ""
    for w in tokens:
        new_text = new_text + " " + stemmer.stem(w)
    return np.char.strip(new_text)

#Converting numbers to its equivalent words
def convert_numbers(data):
    data = np.char.replace(data, "0", " zero ")
    data = np.char.replace(data, "1", " one ")
    data = np.char.replace(data, "2", " two ")
    data = np.char.replace(data, "3", " three ")
    data = np.char.replace(data, "4", " four ")
    data = np.char.replace(data, "5", " five ")
    data = np.char.replace(data, "6", " six ")
    data = np.char.replace(data, "7", " seven ")
    data = np.char.replace(data, "8", " eight ")
    data = np.char.replace(data, "9", " nine ")
    return data

#Removing header
def remove_header(data):
    try:
        ind = data.index('\n\n')
        data = data[ind:]
    except:
        print("No Header")
    return data
```

```
#Removing apostrophe
def remove_apostrophe(data):
    return np.char.replace(data, "'", "")

#Removing single characters
def remove_single_characters(data):
    words = word_tokenize(str(data))
    new_text = ""
    for w in words:
        if len(w) > 1:
            new_text = new_text + " " + w
    return np.char.strip(new_text)
```

:[25] In

```
def preprocess(data, query):
    if not query:
        data = remove_header(data)
        data = convert_lower_case(data)
        data = convert_numbers(data)
        data = remove_punctuation(data)
        data = remove_stop_words(data)
        data = remove_apostrophe(data)
        data = remove_single_characters(data)
        data = stemming(data)
    return data
```

:[26] In

```
postings = pd.DataFrame()
frequency = pd.DataFrame()
```

:[27] In

```
doc = 0
for path in paths:
    file = open(path, 'r', encoding='cp1250')
    text = file.read().strip()
    file.close()
    preprocessed_text = preprocess(text, False)
    if doc%100 == 0:
        print(doc)
    tokens = word_tokenize(str(preprocessed_text))

    pos = 0
    for token in tokens:
        if token in postings:
            p = postings[token][0]
            k = [a[0] for a in p]
            if doc in k:
                for a in p:
                    if a[0] == doc:
                        a[1].add(pos)
            else:
                p.append([doc,{pos}])
                frequency[token][0] += 1
        else:
            postings.insert(value=[[[doc, {pos}]]], loc=0, column=token)
            frequency.insert(value=[1], loc=0, column=token)

        pos += 1
    doc += 1
```

```
0
```

:[28] In

```
postings
```

Out[28]:

call	sipp	simpl	...	truetyp	pete	ventur	undercut	nasti	shop	smirk	rip	lazer	fontmong	
,0]] ,[{7} ,5] [{43} ,10] [{78} ,17] 378} ...	,0]] ,8} [[{29	,0]] ,[{9} ,1] ,[{47} ,15] ,[{8} ,17] ,1080} ...4	...	,99]] ,76 ,57} [[{31	,99]] [[{38}	,99]] [[{60}	,99]] [[{61}	,99]] [[{64}	,99]] [[{71}	,99]] [[{72}	,99]] [[{81}	,99]] [[{92}	[[{98} ,99]]	0

rows × 4821 columns 1

:[29] In

```
frequency
```

Out[29]:

```
ll  sipp  simpl  ...  truetyp  pete  ventur  undercut  nasti  shop  smirk  rip  lazer  fontmong
9      1      7  ...        1     1       1         1      1     1      1    1      1         1  0

                                                                       rows × 4821 columns 1
```

:[30] In

```
postings.to_pickle(title + "_positional_postings")
frequency.to_pickle(title + "_positional_frequency")
```

:[31] In

```
#Read the stored posting list:
postings = pd.read_pickle(title + "_positional_postings")
#Read frequency of a term as follows:
frequency = pd.read_pickle(title + "_positional_frequency")
```

```
def get_word_postings(word):
    preprocessed_word = str(preprocess(word, True))
    print(preprocessed_word)
    print("Frequency:",frequency[preprocessed_word][0])
    print("Postings List:",postings[preprocessed_word][0])

def get_positions(posting_values, doc):
    for posting_value in posting_values:
        if posting_value[0] == doc:
            return posting_value[1]
    return {}

def gen_init_set_matchings(word):
    init = []
    word_postings = postings[word][0]
    for word_posting in word_postings:
        for positions in word_posting[1]:
            init.append((word_posting[0], positions))
    return init

def match_positional_index(init, b):
    matched_docs = []
    for p in init:
        doc = p[0]
        pos = p[1]

        count = 0

        for k in b:
            pos = pos+1
            k_pos = postings[k][0]
            docs_list = [z[0] for z in k_pos]
            if doc in docs_list:
                doc_positions = get_positions(k_pos, doc)
                if pos in doc_positions:
                    count += 1
                else:
                    count += 1
                    break

            if count == len(b):
                matched_docs.append(p[0])
    return set(matched_docs)

def run_query(query):
    processed_query = preprocess(query, True)
    print(processed_query)

    query_tokens = word_tokenize(str(processed_query))
    print(query_tokens)

    if len(query_tokens)==1:
        print("Total Document Mathces", [a[0] for a in postings[query][0]])
        return [a[0] for a in postings[query][0]]

    init_word = query_tokens[0]
```

:[33] In

```
    init_matches = gen_init_set_matchings(init_word)

    query_tokens.pop(0)
    total_matched_docs = match_positional_index(init_matches, query_tokens)
    print("Total Document Matches:", total_matched_docs)
    return total_matched_docs
```

In [34]:

```
get_word_postings("call")
```

```
call
Frequency: 9
Postings List: [[0, {7}], [5, {43}], [10, {78}], [17, {5378, 3204, 393, 272, 11
5, 4566, 3159}], [18, {38}], [64, {896, 5640, 7055, 4626, 4631, 151, 1433, 463
 5, 3755, 1970, 6203, 187, 5202, 4436, 6620, 4446, 864, 6884, 5868, 6775, 5624,
[[{764}], [66, {92}], [74, {34, 4}], [91, {3398, 4431, 6290, 3381, 4031
```

In [41]:

```
list = run_query("hello")
```

```
hello
['hello']
[Total Document Mathces [1, 41, 61, 77, 95
```

In [ ]:

```
```