~~Name:Viviyan Richards W~~

~~D no:205229133~~

In [2]:
```
import nltk
```

In [3]:
```
nltk.download('wordnet')
nltk.download('punkt')
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Angelan\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Angelan\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[3]:
```
True
```

In [5]:
```
text = "This is Andrew's text, isn't it?"
```

1. How many tokens are there if you use WhitespaceTokenizer?. Print tokens.

In [6]:
```
tokenizer = nltk.tokenize.WhitespaceTokenizer()
tokens = tokenizer.tokenize(text)
print(len(tokens))
print(tokens)
```

```
6
['This', 'is', "Andrew's", 'text,', "isn't", 'it?']
```

2 . How many tokens are there if you use TreebankWordTokenizer?. Print tokens.

In [7]:
```
tokenizer = nltk.tokenize.TreebankWordTokenizer()
tokens = tokenizer.tokenize(text)
print(len(tokens))
print(tokens)
```

```
10
['This', 'is', 'Andrew', "'s", 'text', ',', 'is', "n't", 'it', '?']
```

3 . How many tokens there are if you use WordPunctTokenizer?. Print tokens.

In [8]:
```
tokenizer = nltk.tokenize.WordPunctTokenizer()
tokens = tokenizer.tokenize(text)
print(len(tokens))
print(tokens)
```

```
12
['This', 'is', 'Andrew', "'", 's', 'text', ',', 'isn', "'", 't', 'it', '?']
```

EXERCISE-2

1. Open the file: O. Henry's The Gift of the Magi (gift-of-magi.txt).

In [9]:
```
import re
f = open("gift-of-magi.txt", encoding='utf-8')
con=f.read()
print(con)
```

```
The Gift of the Magi
by O. Henry

One dollar and eighty-seven cents. That was all. And sixty cents of it was in
pennies. Pennies saved one and two at a time by bulldozing the grocer and the
vegetable man and the butcher until one's cheeks burned with the silent imput
ation of parsimony that such close dealing implied. Three times Della counted
it. One dollar and eighty-seven cents. And the next day would be Christmas.

There was clearly nothing left to do but flop down on the shabby little couch
and howl. So Della did it. Which instigates the moral reflection that life is
made up of sobs, sniffles, and smiles, with sniffles predominating.

While the mistress of the home is gradually subsiding from the first stage to
the second, take a look at the home. A furnished flat at $8 per week. It did
not exactly beggar description, but it certainly had that word on the look-ou
t for the mendicancy squad.

In the vestibule below was a letter-box into which no letter would go, and an
```

2 . Write a Python script to print out the following:

1. How many word tokens there are

In [10]:
```
tokenizer = nltk.tokenize.WhitespaceTokenizer()
tokens = tokenizer.tokenize(con)
print(len(tokens))
```

```
2074
```

2 . How many word types there are, (word types are a unique set of words)

In [11]:
```
from nltk import *
data=FreqDist(tokens)
data
```

Out[11]:
```
FreqDist({'the': 107, 'and': 74, 'a': 64, 'of': 51, 'to': 41, 'was': 26, 'she':
25, 'in': 24, 'had': 21, 'her': 21, ...})
```

3 . Top 20 most frequent words and their counts

In [24]:
```
data.most_common(20)
```

Out[24]:
```
[('the', 107),
 ('and', 74),
 ('a', 64),
 ('of', 51),
 ('to', 41),
 ('was', 26),
 ('she', 25),
 ('in', 24),
 ('had', 21),
 ('her', 21),
 ('that', 20),
 ('it', 19),
 ('at', 19),
 ('with', 19),
 ('for', 19),
 ('his', 17),
 ('on', 16),
 ('I', 14),
 ('Jim', 13),
 ('were', 11)]
```

4 . Words that are at least 10 characters long and their counts

In [22]:
```
from nltk import *
text=[w for w in tokens if len(w)>10]
print(text)
freq=FreqDist(text)
freq
```

```
['eighty-seven', 'eighty-seven', 'predominating.', 'description,', 'appertainin
g', '"Dillingham"', '"Dillingham"', 'contracting', 'calculated.', 'sterling--so
mething', 'longitudinal', 'brilliantly,', 'possessions', "grandfather's.", '"So
fronie."', 'proclaiming', 'meretricious', 'ornamentation--as', 'description',
'intoxication', 'close-lying', 'wonderfully', 'critically.', 'eighty-seven', 't
wenty-two--and', 'disapproval,', "Christmas!'", 'laboriously,', 'inconsequentia
l', 'difference?', 'mathematician', 'illuminated', 'necessitating', 'tortoise-s
hell,', 'possession.', 'men--wonderfully', 'duplication.']
```

Out[22]:
```
FreqDist({'eighty-seven': 3, '"Dillingham"': 2, 'predominating.': 1, 'descripti
on,': 1, 'appertaining': 1, 'contracting': 1, 'calculated.': 1, 'sterling--some
thing': 1, 'longitudinal': 1, 'brilliantly,': 1, ...})
```

5 . 10+ characters-long words that occur at least twice, sorted from most frequent to least

In [23]:
```
text = [w for w in tokens if len(w)>10]
s=FreqDist(text)
s
```

Out[23]:
```
FreqDist({'eighty-seven': 3, '"Dillingham"': 2, 'predominating.': 1, 'description,': 1, 'appertaining': 1, 'contracting': 1, 'calculated.': 1, 'sterling--something': 1, 'longitudinal': 1, 'brilliantly,': 1, ...})
```

In [28]:
```
for i,j in freq.items():
    if len(i) > 10 and j>2:
        print(i,j)
```

```
eighty-seven 3
```

EXERCISE -3:

List Comprehension

STEP-1

In [38]:
```
fname = "./data/austen-emma.txt"
f = open("austen-emma.txt", encoding='utf-8')
etxt=f.read()
f.close()
```

In [39]:
```
etxt[-200:]
```

Out[39]:
```
'e deficiencies, the wishes,\nthe hopes, the confidence, the predictions of the small band\nof true friends who witnessed the ceremony, were fully answered\nin the perfect happiness of the union.\n\n\nFINIS\n'
```

In [40]:
```
tokenizer = nltk.tokenize.WhitespaceTokenizer()
tokens = tokenizer.tokenize(etxt)
tokens[-20:]
```

Out[40]:
```
['small',
 'band',
 'of',
 'true',
 'friends',
 'who',
 'witnessed',
 'the',
 'ceremony,',
 'were',
 'fully',
 'answered',
 'in',
 'the',
 'perfect',
 'happiness',
 'of',
 'the',
 'union.',
 'FINIS']
```

In [41]:
```
etoks = nltk.word_tokenize(etxt.lower())
etoks[-20:]
```

Out[41]:
```
['of',
 'true',
 'friends',
 'who',
 'witnessed',
 'the',
 'ceremony',
 ',',
 'were',
 'fully',
 'answered',
 'in',
 'the',
 'perfect',
 'happiness',
 'of',
 'the',
 'union',
 '.',
 'finis']
```

In [42]:
```
len(etoks)
```

Out[42]:
```
191781
```

In [43]:
```
etypes=sorted(set(etoks))
```

In [44]:
```
etypes[-10:]
```

Out[44]:
```
['younger',
 'youngest',
 'your',
 'yours',
 'yourself',
 'yourself.',
 'youth',
 'youthful',
 'zeal',
 'zigzags']
```

In [45]:
```
len(etypes)
```

Out[45]:
```
7944
```

In [46]:
```
efreq = nltk.FreqDist(etoks)
```

In [47]:
```
efreq['beautiful']
```

Out[47]:
```
24
```

STEP 2: list-comprehend Emma

In [48]:
```
etxt
```

Out[48]:
```
'[Emma by Jane Austen 1816]\n\nVOLUME I\n\nCHAPTER I\n\n\nEmma Woodhouse, han
dsome, clever, and rich, with a comfortable home\nand happy disposition, seem
ed to unite some of the best blessings\nof existence; and had lived nearly tw
enty-one years in the world\nwith very little to distress or vex her.\n\nShe
was the youngest of the two daughters of a most affectionate,\nindulgent fath
er; and had, in consequence of her sister\'s marriage,\nbeen mistress of his
house from a very early period.  Her mother\nhad died too long ago for her to
have more than an indistinct\nremembrance of her caresses; and her place had
been supplied\nby an excellent woman as governess, who had fallen little shor
t\nof a mother in affection.\n\nSixteen years had Miss Taylor been in Mr. Woo
dhouse\'s family,\nless as a governess than a friend, very fond of both daugh
ters,\nbut particularly of Emma.  Between _them_ it was more the intimacy\nof
sisters.  Even before Miss Taylor had ceased to hold the nominal\noffice of g
overness, the mildness of her temper had hardly allowed\nher to impose any re
straint; and the shadow of authority being\nnow long passed away, they had be
en living together as friend and\nfriend very mutually attached, and Emma doi
ng just what she liked;\nhighly esteeming Miss Taylor\'s judgment, but direct
ed chiefly by\nher own.\n\nThe real evils, indeed, of Emma\'s situation were
the power of having\nrather too much her own way, and a disposition to think
```

Question 1: Words with prefix and suffix

What are the words that start with 'un' and end in 'able'?

In [49]:
```
[word for word in tokens if word.startswith("un") & word.endswith("able")]
```

Out[49]:
```
['unexceptionable',
 'unsuitable',
 'unreasonable',
 'unreasonable',
 'uncomfortable',
 'unfavourable',
 'unexceptionable',
 'uncomfortable',
 'unpersuadable',
 'unavoidable',
 'unsuitable',
 'unmanageable',
 'unreasonable',
 'unobjectionable',
 'unpersuadable',
 'unexceptionable',
 'unpardonable',
 'unmanageable',
 'unfavourable',
 'unaccountable',
 'unable',
 'unable',
 'unpardonable',
 'unexceptionable',
 'unreasonable',
 'unreasonable',
 'unpardonable',
 'unexceptionable',
 'unreasonable']
```

Question 2: Length

How many Emma word types are 15 characters or longer? Exclude hyphenated words.

In [50]:
```
tokenizer = nltk.tokenize.WordPunctTokenizer()
toke= tokenizer.tokenize(etxt)
```

In [51]:
```
[word for word in toke if len(word)>15]
```

Out[51]:
```
['companionableness',
 'misunderstanding',
 'incomprehensible',
 'undistinguishing',
 'unceremoniousness',
 'Disingenuousness',
 'disagreeableness',
 'misunderstandings',
 'misunderstandings',
 'misunderstandings',
 'misunderstandings',
 'disinterestedness',
 'unseasonableness']
```

Average word length

What's the average length of all Emma word types?

In [53]:
```
average=sum(len(word)for word in toke)/len(toke)
average
```

Out[53]:
```
3.755268231589122
```

In [54]:
```
lg = []
for i in toke:
    if len(i)>15:
        lg.append(i)
print(lg)
```

```
['companionableness', 'misunderstanding', 'incomprehensible', 'undistinguishin
g', 'unceremoniousness', 'Disingenuousness', 'disagreeableness', 'misunderstand
ings', 'misunderstandings', 'misunderstandings', 'misunderstandings', 'disinter
estedness', 'unseasonableness']
```

Question 4: Word frequency

How many Emma word types have a frequency count of 200 or more?

In [57]:
```
from nltk import *
fdiemm = FreqDist(toke)
```

In [59]:
```
for i,j in fdiemm.items():
    if j > 200:
        print(i,j)
```

```
your 337
sure 204
will 559
are 447
You 303
may 213
me 564
do 580
about 246
Knightley 389
out 212
quite 269
," 421
has 243
should 366
can 270
nothing 237
Elton 385
Churchill 223
Frank 208
```

How many word types appear only once?

In [60]:
```
for i,j in fdiemm.items():
    if j == 1:
        print(i,j)
```

```
Austen 1
1816 1
] 1
vex 1
indistinct 1
caresses 1
nominal 1
mildness 1
impose 1
esteeming 1
disadvantages 1
misfortunes 1
Sorrow 1
mournful 1
debt 1
tenderer 1
valetudinarian 1
amounting 1
equals 1
```

STEP 3: bigrams in Emma

Question 6: Bigrams

What are the last 10 bigrams

In [62]:
```
e2grams = list(nltk.bigrams(toke))
e2gramfd = nltk.FreqDist(e2grams)
```

In [63]:
```
e2gramfd
```

Out[63]:
```
FreqDist({(',', 'and'): 1879, ('Mr', '.'): 1153, ("'", 's'): 932, (';', 'and'): 866, ('."', '"'): 757, ('Mrs', '.'): 699, ('to', 'be'): 595, ('.', 'I'): 570, (',', 'I'): 568, ('of', 'the'): 556, ...})
```

In [64]:
```
last_ten = FreqDist(dict(e2gramfd.most_common()[-10:]))
last_ten
```

Out[64]:
```
FreqDist({('who', 'witnessed'): 1, ('witnessed', 'the'): 1, ('the', 'ceremon
y'): 1, ('were', 'fully'): 1, ('fully', 'answered'): 1, ('answered', 'in'): 1,
('the', 'perfect'): 1, ('the', 'union'): 1, ('union', '.'): 1, ('.', 'FINIS'):
1})
```

Question 7: Bigram top frequency

What are the top 20 most frequent bigrams?

In [65]:
```
tokenizer = nltk.tokenize.WhitespaceTokenizer()
tokes = tokenizer.tokenize(etxt)
```

In [66]:
```
e2grams = list(nltk.bigrams(tokes))
e2gramfd = nltk.FreqDist(e2grams)
```

In [67]:
```
e2gramfd.most_common(20)
```

Out[67]:
```
[(('to', 'be'), 562),
 (('of', 'the'), 556),
 (('in', 'the'), 431),
 (('I', 'am'), 302),
 (('had', 'been'), 299),
 (('could', 'not'), 270),
 (('it', 'was'), 253),
 (('she', 'had'), 242),
 (('to', 'the'), 236),
 (('have', 'been'), 233),
 (('of', 'her'), 230),
 (('I', 'have'), 214),
 (('and', 'the'), 208),
 (('would', 'be'), 208),
 (('she', 'was'), 206),
 (('do', 'not'), 196),
 (('of', 'his'), 182),
 (('that', 'she'), 178),
 (('to', 'have'), 176),
 (('such', 'a'), 176)]
```

Question 8: Bigram frequency count

#How many times does the bigram 'so happy' appear?

In [68]:
```
for i , j in e2gramfd.items():
    if i == ('so', 'happy'):
        print(i,j)
```

```
('so', 'happy') 3
```

Question 9: Word following 'so'

What are the words that follow 'so'? What are their frequency counts? (For loop will be easier; see if you can utilize list comprehension for this.)

In [69]:
```
import re
from collections import Counter
```

In [70]:
```
words = re.findall(r'so+ \w+',open('austen-emma.txt').read())
ab = Counter(zip(words))
print(ab)
```

```
Counter({('so much',): 95, ('so very',): 76, ('so well',): 30, ('so many',): 2
7, ('so long',): 27, ('so little',): 20, ('so far',): 17, ('so I',): 14, ('so k
ind',): 13, ('so good',): 12, ('so often',): 10, ('so soon',): 9, ('so grea
t',): 8, ('so to',): 7, ('so fond',): 7, ('so she',): 7, ('so it',): 6, ('so an
xious',): 6, ('so as',): 6, ('so you',): 6, ('so truly',): 6, ('so completel
y',): 5, ('so obliging',): 5, ('so extremely',): 5, ('so entirely',): 4, ('so h
appy',): 4, ('so interesting',): 4, ('so fast',): 4, ('so near',): 4, ('so plea
sed',): 4, ('so few',): 4, ('so that',): 4, ('so strong',): 4, ('so liberal',):
4, ('so miserable',): 4, ('so happily',): 3, ('so proper',): 3, ('so pleasantl
y',): 3, ('so superior',): 3, ('so warmly',): 3, ('so bad',): 3, ('so odd',):
3, ('so ill',): 3, ('so delighted',): 3, ('so particularly',): 3, ('so easil
y',): 3, ('so on',): 3, ('so attentive',): 3, ('so fortunate',): 3, ('so gla
d',): 3, ('so shocked',): 3, ('so at',): 3, ('so obliged',): 2, ('so perfectl
y',): 2, ('so dear',): 2, ('so busy',): 2, ('so did',): 2, ('so forth',): 2,
('so totally',): 2, ('so remarkably',): 2, ('so plainly',): 2, ('so charmin
g',): 2, ('so surprized',): 2, ('so early',): 2, ('so too',): 2, ('so easy',):
2, ('so decidedly',): 2, ('so absolutely',): 2, ('so particular',): 2, ('so dec
eived',): 2, ('so palpably',): 2, ('so clever',): 2, ('so short',): 2, ('so col
d',): 2, ('so high',): 2, ('so happened',): 2, ('so full',): 2, ('so thoroughl
y',): 2, ('so equal',): 2, ('so off',): 2, ('so naturally',): 2, ('so afrai
d',): 2, ('so deep',): 2, ('so kindly',): 2, ('so pale',): 2, ('so noble',): 2,
('so lovely',): 2, ('so mad',): 2, ('so nearly',): 2, ('so sorry',): 2, ('so ch
eerful',): 2, ('so unfeeling',): 2, ('so ready',): 2, ('so unperceived',): 1,
('so mild',): 1, ('so constantly',): 1, ('so comfortably',): 1, ('so avowed',):
1, ('so deservedly',): 1, ('so convenient',): 1, ('so just',): 1, ('so apparen
t',): 1, ('so sorrowful',): 1, ('so spent',): 1, ('so artlessly',): 1, ('so pla
in',): 1, ('so firmly',): 1, ('so genteel',): 1, ('so _then_',): 1, ('so brilli
ant',): 1, ('so seldom',): 1, ('so nervous',): 1, ('so indeed',): 1, ('so pac
k',): 1, ('so doubtful',): 1, ('so with',): 1, ('so contemptible',): 1, ('so sl
ightingly',): 1, ('so by',): 1, ('so loudly',): 1, ('so materially',): 1, ('so
hard',): 1, ('so delightful',): 1, ('so pointed',): 1, ('so equalled',): 1, ('s
o evidently',): 1, ('so immediately',): 1, ('so sought',): 1, ('so excellen
t',): 1, ('so prettily',): 1, ('so extreme',): 1, ('so wonder',): 1, ('so alway
s',): 1, ('so silly',): 1, ('so satisfied',): 1, ('so smiling',): 1, ('so prosi
ng',): 1, ('so undistinguishing',): 1, ('so apt',): 1, ('so dreadful',): 1, ('s
o respected',): 1, ('so tenderly',): 1, ('so grieved',): 1, ('so shocking',):
1, ('so conceited',): 1, ('so before',): 1, ('so prevalent',): 1, ('so heav
y',): 1, ('so swiftly',): 1, ('so spoken',): 1, ('so or',): 1, ('so overcharge
d',): 1, ('so pleasant',): 1, ('so fenced',): 1, ('so hospitable',): 1, ('so in
terested',): 1, ('so sanguine',): 1, ('so sure',): 1, ('so careless',): 1, ('so
rapidly',): 1, ('so frequent',): 1, ('so sensible',): 1, ('so misled',): 1, ('s
o blind',): 1, ('so complaisant',): 1, ('so misinterpreted',): 1, ('so activ
e',): 1, ('so pointedly',): 1, ('so striking',): 1, ('so sudden',): 1, ('so ind
ustriously',): 1, ('so partial',): 1, ('so natural',): 1, ('so inevitable',):
1, ('so lately',): 1, ('so beautifully',): 1, ('so distinct',): 1, ('so conside
rate',): 1, ('so light',): 1, ('so intimate',): 1, ('so magnified',): 1, ('so c
autious',): 1, ('so confined',): 1, ('so wish',): 1, ('so he',): 1, ('so glorio
us',): 1, ('so quick',): 1, ('so sweetly',): 1, ('so inseparably',): 1, ('so de
serving',): 1, ('so disappointed',): 1, ('so ended',): 1, ('so sluggish',): 1,
('so amiable',): 1, ('so quiet',): 1, ('so idolized',): 1, ('so cried',): 1,
('so acceptable',): 1, ('so properly',): 1, ('so reasonable',): 1, ('so delight
fully',): 1, ('so rich',): 1, ('so warm',): 1, ('so large',): 1, ('so handsomel
```

```
y',): 1, ('so abundant',): 1, ('so outree',): 1, ('so thoughtful',): 1, ('so mu
st',): 1, ('so effectually',): 1, ('so beautiful',): 1, ('so Patty',): 1, ('so
honoured',): 1, ('so close',): 1, ('so imprudent',): 1, ('so limited',): 1, ('s
o from',): 1, ('so amusing',): 1, ('so indifferent',): 1, ('so indignant',): 1,
('so said',): 1, ('so right',): 1, ('so wretched',): 1, ('so now',): 1, ('so oc
cupied',): 1, ('so unhappy',): 1, ('so highly',): 1, ('so generally',): 1, ('so
exactly',): 1, ('so double',): 1, ('so secluded',): 1, ('so regular',): 1, ('so
determined',): 1, ('so motherly',): 1, ('so the',): 1, ('so glibly',): 1, ('so
calculated',): 1, ('so thrown',): 1, ('so exclusively',): 1, ('so disgustingl
y',): 1, ('so needlessly',): 1, ('so does',): 1, ('so resolutely',): 1, ('so wo
uld',): 1, ('so infinitely',): 1, ('so fluently',): 1, ('so they',): 1, ('so im
patient',): 1, ('so briskly',): 1, ('so vigorously',): 1, ('so young',): 1, ('s
o hardened',): 1, ('so gratified',): 1, ('so received',): 1, ('so then',): 1,
('so and',): 1, ('so gratefully',): 1, ('so found',): 1, ('so placed',): 1, ('s
o lain',): 1, ('so his',): 1, ('so arranged',): 1, ('so moving',): 1, ('so walk
ing',): 1, ('so when',): 1, ('so favourable',): 1, ('so late',): 1, ('so silen
t',): 1, ('so dull',): 1, ('so irksome',): 1, ('so agitated',): 1, ('so bruta
l',): 1, ('so cruel',): 1, ('so depressed',): 1, ('so no',): 1, ('so justly',):
1, ('so astonished',): 1, ('so will',): 1, ('so simple',): 1, ('so dignifie
d',): 1, ('so suddenly',): 1, ('so a',): 1, ('so herself',): 1, ('so peremptori
ly',): 1, ('so uneasy',): 1, ('so wonderful',): 1, ('so _very_',): 1, ('so expr
essly',): 1, ('so angry',): 1, ('so anxiously',): 1, ('so strange',): 1, ('so s
toutly',): 1, ('so mistake',): 1, ('so mistaken',): 1, ('so dreadfully',): 1,
('so voluntarily',): 1, ('so satisfactory',): 1, ('so disinterested',): 1, ('so
foolishly',): 1, ('so ingeniously',): 1, ('so entreated',): 1, ('so like',): 1,
('so cordially',): 1, ('so essential',): 1, ('so designedly',): 1, ('so hast
y',): 1, ('so richly',): 1, ('so grateful',): 1, ('so tenaciously',): 1, ('so f
eeling',): 1, ('so engaging',): 1, ('so engaged',): 1, ('so hot',): 1, ('so use
ful',): 1, ('so attached',): 1, ('so peculiarly',): 1, ('so singularly',): 1,
('so taken',): 1, ('so recently',): 1, ('so fresh',): 1, ('so hateful',): 1,
('so heartily',): 1, ('so steady',): 1, ('so complete',): 1, ('so in',): 1, ('s
o suffered',): 1})
```

Question 10: Trigrams¶

What are the last 10 trigrams

In [71]:
```
e3grams = list(nltk.trigrams(tokes))
e3gramfd = nltk.FreqDist(e3grams)
```

In [72]:
```
last_ten = FreqDist(dict(e3gramfd.most_common()[-10:]))
last_ten
```

Out[72]:
```
FreqDist({('the', 'ceremony,', 'were'): 1, ('ceremony,', 'were', 'fully'): 1,
('were', 'fully', 'answered'): 1, ('fully', 'answered', 'in'): 1, ('answered',
'in', 'the'): 1, ('in', 'the', 'perfect'): 1, ('the', 'perfect', 'happiness'):
1, ('perfect', 'happiness', 'of'): 1, ('of', 'the', 'union.'): 1, ('the', 'unio
n.', 'FINIS'): 1})
```

Question 11: Trigram top frequency

What are the top 10 most frequent trigrams?

In [74]:
```
e3gramfd.most_common(10)
```

Out[74]:
```
[(('I', 'do', 'not'), 94),
 (('I', 'am', 'sure'), 75),
 (('would', 'have', 'been'), 55),
 (('a', 'great', 'deal'), 55),
 (('she', 'could', 'not'), 49),
 (('could', 'not', 'be'), 45),
 (('she', 'had', 'been'), 44),
 (('it', 'would', 'be'), 43),
 (('do', 'not', 'know'), 43),
 (('Mr.', 'and', 'Mrs.'), 37)]
```

Question 12: Trigram frequency count

How many times does the trigram 'so happy to' appear?

In [78]:
```
for i , j in e3gramfd.items():
    if i == ('so', 'happy','to'):
        print(i,j)
```

In [ ]:

Name:Viviyan Richards W

Roll no:205229133

Lab2. Computing Bigram Frequencies

EXERCISE-1: Process simple bigram data file

STEP 1: OPEN the file, count_2w.txt

In [1]:
```
import io
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
with io.open('count_2w.txt','r',encoding='utf8') as f:
    text = f.readlines()
```

STEP 2: build goog2w_list

In [3]:
```
mini = text[:10]
```

In [4]:
```
nimi = text[:]
```

In [5]:
```
mini[0].split()
```

Out[5]:
```
['0Uplink', 'verified', '523545']
```

In [6]:
```
mini_list = []
for m in mini:
    (w1, w2, count) = m.split()
    count = int(count)
    mini_list.append(((w1, w2), count))
mini_list
```

Out[6]:
```
[(('0Uplink', 'verified'), 523545),
 (('0km', 'to'), 116103),
 (('1000s', 'of'), 939476),
 (('100s', 'of'), 539389),
 (('100th', 'anniversary'), 158621),
 (('10am', 'to'), 376141),
 (('10th', 'and'), 183715),
 (('10th', 'anniversary'), 242830),
 (('10th', 'century'), 117755),
 (('10th', 'grade'), 174046)]
```

In [7]:
```
mini_list[0]
```

Out[7]:
```
(('0Uplink', 'verified'), 523545)
```

In [8]:
```
goog2w_list = []
for m in nimi:
    (w1, w2, count) = m.split()
    count = int(count)
    goog2w_list.append(((w1, w2), count))
goog2w_list
```

Out[8]:
```
[(('0Uplink', 'verified'), 523545),
 (('0km', 'to'), 116103),
 (('1000s', 'of'), 939476),
 (('100s', 'of'), 539389),
 (('100th', 'anniversary'), 158621),
 (('10am', 'to'), 376141),
 (('10th', 'and'), 183715),
 (('10th', 'anniversary'), 242830),
 (('10th', 'century'), 117755),
 (('10th', 'grade'), 174046),
 (('10th', 'in'), 107194),
 (('10th', 'of'), 277970),
 (('11am', 'to'), 127624),
 (('11th', 'and'), 178884),
 (('11th', 'century'), 168601),
 (('11th', 'grade'), 126301),
 (('11th', 'of'), 189501),
 (('125Mbps', 'w'), 108645),
 (('12th', 'and'), 136706),
```

In [9]:
```
goog2w_list[0]
```

Out[9]:
```
(('0Uplink', 'verified'), 523545)
```

STEP 3: build goog2w_fd

In [10]:
```
!pip install nltk
```

```
Requirement already satisfied: nltk in c:\programdata\anaconda3\lib\site-packag
es (3.5)
Requirement already satisfied: joblib in c:\programdata\anaconda3\lib\site-pack
ages (from nltk) (0.17.0)
Requirement already satisfied: tqdm in c:\programdata\anaconda3\lib\site-packag
es (from nltk) (4.50.2)
Requirement already satisfied: click in c:\programdata\anaconda3\lib\site-packa
ges (from nltk) (7.1.2)
Requirement already satisfied: regex in c:\programdata\anaconda3\lib\site-packa
ges (from nltk) (2020.10.15)
```

In [11]:
```
import nltk
goog2w_fd = nltk.FreqDist()
goog2w_fd
```

Out[11]:
```
FreqDist({})
```

In [12]:
```
for m in text:
    w1, w2, count = m.split()
    goog2w_fd[(w1, w2)] = count
```

In [13]:
```
goog2w_fd[('of', 'the')]
```

Out[13]:
```
'2766332391'
```

In [14]:
```
goog2w_fd[('so', 'beautiful')]
```

Out[14]:
```
'612472'
```

STEP 4: explore

1. What are the top-10 bigrams?

In [15]:
```
goog2w_fd.most_common(10)
```

Out[15]:
```
[(('You', 'think'), '999988'),
 (('a', 'middle'), '999987'),
 (('his', 'wife'), '9999448'),
 (('traditional', 'and'), '999927'),
 (('Ask', 'your'), '999907'),
 (('towards', 'the'), '9998989'),
 (('<S>', 'central'), '999848'),
 (('no', 'man'), '999833'),
 (('committee', 'members'), '999819'),
 (('each', 'country'), '999818')]
```

STEP 5: pickle the data

In [16]:
```
import pickle as pkl
```

In [17]:
```
with open('goog2w_list.pkl', 'ab') as handle:
    pkl.dump(goog2w_list,handle)
```

In [18]:
```
with open('goog2w_fd.pkl', 'ab') as handle:
    pkl.dump(goog2w_fd,handle)
```

EXERCISE - 2 Frequency distribution from Jane Austen Novels

A. opens (and later closes) the text file, reads in the string content,

In [19]:
```
with open('austen-emma.txt','r') as fl:
    cona=fl.read()
```

In [20]:
```
with open('austen-persuasion.txt','r') as flp:
    conp=flp.read()
```

In [21]:
```
with open('austen-sense.txt','r') as fls:
    cons=fls.read()
```

B. builds a list of individual sentences,

In [22]:
```
from nltk.tokenize import sent_tokenize as st
```

In [23]:
```
st(cona)
```

Out[23]:
```
['[Emma by Jane Austen 1816]\n\nVOLUME I\n\nCHAPTER I\n\n\nEmma Woodhouse, ha
ndsome, clever, and rich, with a comfortable home\nand happy disposition, see
med to unite some of the best blessings\nof existence; and had lived nearly t
wenty-one years in the world\nwith very little to distress or vex her.',
 "She was the youngest of the two daughters of a most affectionate,\nindulgen
t father; and had, in consequence of her sister's marriage,\nbeen mistress of
his house from a very early period.",
 'Her mother\nhad died too long ago for her to have more than an indistinct\n
remembrance of her caresses; and her place had been supplied\nby an excellent
woman as governess, who had fallen little short\nof a mother in affection.',
 "Sixteen years had Miss Taylor been in Mr. Woodhouse's family,\nless as a go
verness than a friend, very fond of both daughters,\nbut particularly of Emm
a.",
 'Between _them_ it was more the intimacy\nof sisters.',
 "Even before Miss Taylor had ceased to hold the nominal\noffice of governes
s, the mildness of her temper had hardly allowed\nher to impose any restrain
t; and the shadow of authority being\nnow long passed away, they had been liv
ing together as friend and\nfriend very mutually attached, and Emma doing jus
t what she liked;\nhighly esteeming Miss Taylor's judgment, but directed chie
```

In [24]:
```
st(conp)
```

Out[24]:
```
['[Persuasion by Jane Austen 1818]\n\n\nChapter 1\n\n\nSir Walter Elliot, of
Kellynch Hall, in Somersetshire, was a man who,\nfor his own amusement, never
took up any book but the Baronetage;\nthere he found occupation for an idle h
our, and consolation in a\ndistressed one; there his faculties were roused in
to admiration and\nrespect, by contemplating the limited remnant of the earli
est patents;\nthere any unwelcome sensations, arising from domestic affairs\n
changed naturally into pity and contempt as he turned over\nthe almost endles
s creations of the last century; and there,\nif every other leaf were powerle
ss, he could read his own history\nwith an interest which never failed.',
 'This was the page at which\nthe favourite volume always opened:\n\n
"ELLIOT OF KELLYNCH HALL.',
 '"Walter Elliot, born March 1, 1760, married, July 15, 1784, Elizabeth,\ndau
ghter of James Stevenson, Esq.',
 'of South Park, in the county of\nGloucester, by which lady (who died 1800)
he has issue Elizabeth,\nborn June 1, 1785; Anne, born August 9, 1787; a stil
l-born son,\nNovember 5, 1789; Mary, born November 20, 1791."',
 'Precisely such had the paragraph originally stood from the printer\'s hand
s;\nbut Sir Walter had improved it by adding, for the information of\nhimself
and his family, these words, after the date of Mary\'s birth--\n"Married, Dec
```

In [25]:
```
st(cons)
```

Out[25]:
```
['[Sense and Sensibility by Jane Austen 1811]\n\nCHAPTER 1\n\n\nThe family of
Dashwood had long been settled in Sussex.',
 'Their estate was large, and their residence was at Norland Park,\nin the ce
ntre of their property, where, for many generations,\nthey had lived in so re
spectable a manner as to engage\nthe general good opinion of their surroundin
g acquaintance.',
 'The late owner of this estate was a single man, who lived\nto a very advanc
ed age, and who for many years of his life,\nhad a constant companion and hou
sekeeper in his sister.',
 'But her death, which happened ten years before his own,\nproduced a great a
lteration in his home; for to supply\nher loss, he invited and received into
his house the family\nof his nephew Mr. Henry Dashwood, the legal inheritor\n
of the Norland estate, and the person to whom he intended\nto bequeath it.',
 "In the society of his nephew and niece,\nand their children, the old Gentle
man's days were\ncomfortably spent.",
 'His attachment to them all increased.',
 'The constant attention of Mr. and Mrs. Henry Dashwood\nto his wishes, which
proceeded not merely from interest,\nbut from goodness of heart, gave him eve
ry degree of solid\ncomfort which his age could receive; and the cheerfulness
```

C. prints out how many sentences there are,

In [26]:
```
print(len(st(cona)))
print(len(st(conp)))
print(len(st(cons)))
```

```
7493
3654
4833
```

E. prints the token and the type counts of this corpus,

In [27]:
```
from nltk.tokenize import word_tokenize
```

In [28]:
```
t1=word_tokenize(cona)
print(t1)
```

```
['[', 'Emma', 'by', 'Jane', 'Austen', '1816', ']', 'VOLUME', 'I', 'CHAPTER',
'I', 'Emma', 'Woodhouse', ',', 'handsome', ',', 'clever', ',', 'and', 'rich',
',', 'with', 'a', 'comfortable', 'home', 'and', 'happy', 'disposition', ',',
'seemed', 'to', 'unite', 'some', 'of', 'the', 'best', 'blessings', 'of', 'exi
stence', ';', 'and', 'had', 'lived', 'nearly', 'twenty-one', 'years', 'in',
'the', 'world', 'with', 'very', 'little', 'to', 'distress', 'or', 'vex', 'he
r', '.', 'She', 'was', 'the', 'youngest', 'of', 'the', 'two', 'daughters', 'o
f', 'a', 'most', 'affectionate', ',', 'indulgent', 'father', ';', 'and', 'ha
d', ',', 'in', 'consequence', 'of', 'her', 'sister', "'s", 'marriage', ',',
'been', 'mistress', 'of', 'his', 'house', 'from', 'a', 'very', 'early', 'peri
od', '.', 'Her', 'mother', 'had', 'died', 'too', 'long', 'ago', 'for', 'her',
'to', 'have', 'more', 'than', 'an', 'indistinct', 'remembrance', 'of', 'her',
'caresses', ';', 'and', 'her', 'place', 'had', 'been', 'supplied', 'by', 'a
n', 'excellent', 'woman', 'as', 'governess', ',', 'who', 'had', 'fallen', 'li
ttle', 'short', 'of', 'a', 'mother', 'in', 'affection', '.', 'Sixteen', 'year
s', 'had', 'Miss', 'Taylor', 'been', 'in', 'Mr.', 'Woodhouse', "'s", 'famil
y', ',', 'less', 'as', 'a', 'governess', 'than', 'a', 'friend', ',', 'very',
'fond', 'of', 'both', 'daughters', ',', 'but', 'particularly', 'of', 'Emma',
'.', 'Between', '_them_', 'it', 'was', 'more', 'the', 'intimacy', 'of', 'sist
```

In [29]:
```
t2=word_tokenize(conp)
print(t2)
```

```
['[', 'Persuasion', 'by', 'Jane', 'Austen', '1818', ']', 'Chapter', '1', 'Si
r', 'Walter', 'Elliot', ',', 'of', 'Kellynch', 'Hall', ',', 'in', 'Somersetsh
ire', ',', 'was', 'a', 'man', 'who', ',', 'for', 'his', 'own', 'amusement',
',', 'never', 'took', 'up', 'any', 'book', 'but', 'the', 'Baronetage', ';',
'there', 'he', 'found', 'occupation', 'for', 'an', 'idle', 'hour', ',', 'an
d', 'consolation', 'in', 'a', 'distressed', 'one', ';', 'there', 'his', 'facu
lties', 'were', 'roused', 'into', 'admiration', 'and', 'respect', ',', 'by',
'contemplating', 'the', 'limited', 'remnant', 'of', 'the', 'earliest', 'paten
ts', ';', 'there', 'any', 'unwelcome', 'sensations', ',', 'arising', 'from',
'domestic', 'affairs', 'changed', 'naturally', 'into', 'pity', 'and', 'contem
pt', 'as', 'he', 'turned', 'over', 'the', 'almost', 'endless', 'creations',
'of', 'the', 'last', 'century', ';', 'and', 'there', ',', 'if', 'every', 'oth
er', 'leaf', 'were', 'powerless', ',', 'he', 'could', 'read', 'his', 'own',
'history', 'with', 'an', 'interest', 'which', 'never', 'failed', '.', 'This',
'was', 'the', 'page', 'at', 'which', 'the', 'favourite', 'volume', 'always',
'opened', ':', '``', 'ELLIOT', 'OF', 'KELLYNCH', 'HALL', '.', '``', 'Walter',
'Elliot', ',', 'born', 'March', '1', ',', '1760', ',', 'married', ',', 'Jul
y', '15', ',', '1784', ',', 'Elizabeth', ',', 'daughter', 'of', 'James', 'Ste
venson', ',', 'Esq', '.', 'of', 'South', 'Park', ',', 'in', 'the', 'county',
```

In [30]:
```
t3 = word_tokenize(cons)
print(t3)
```

```
['[', 'Sense', 'and', 'Sensibility', 'by', 'Jane', 'Austen', '1811', ']', 'CH
APTER', '1', 'The', 'family', 'of', 'Dashwood', 'had', 'long', 'been', 'settl
ed', 'in', 'Sussex', '.', 'Their', 'estate', 'was', 'large', ',', 'and', 'the
ir', 'residence', 'was', 'at', 'Norland', 'Park', ',', 'in', 'the', 'centre',
'of', 'their', 'property', ',', 'where', ',', 'for', 'many', 'generations',
',', 'they', 'had', 'lived', 'in', 'so', 'respectable', 'a', 'manner', 'as',
'to', 'engage', 'the', 'general', 'good', 'opinion', 'of', 'their', 'surround
ing', 'acquaintance', '.', 'The', 'late', 'owner', 'of', 'this', 'estate', 'w
as', 'a', 'single', 'man', ',', 'who', 'lived', 'to', 'a', 'very', 'advance
d', 'age', ',', 'and', 'who', 'for', 'many', 'years', 'of', 'his', 'life',
',', 'had', 'a', 'constant', 'companion', 'and', 'housekeeper', 'in', 'his',
'sister', '.', 'But', 'her', 'death', ',', 'which', 'happened', 'ten', 'year
s', 'before', 'his', 'own', ',', 'produced', 'a', 'great', 'alteration', 'i
n', 'his', 'home', ';', 'for', 'to', 'supply', 'her', 'loss', ',', 'he', 'inv
ited', 'and', 'received', 'into', 'his', 'house', 'the', 'family', 'of', 'hi
s', 'nephew', 'Mr.', 'Henry', 'Dashwood', ',', 'the', 'legal', 'inheritor',
'of', 'the', 'Norland', 'estate', ',', 'and', 'the', 'person', 'to', 'whom',
'he', 'intended', 'to', 'bequeath', 'it', '.', 'In', 'the', 'society', 'of',
'his', 'nephew', 'and', 'niece', ',', 'and', 'their', 'children', ',', 'the',
```

F. builds a frequency count dictionary of words,

In [31]:
```
from nltk import *
```

In [32]:
```
da1 = FreqDist(t1)
da1
```

Out[32]:
```
FreqDist({',': 12016, '.': 6355, 'to': 5125, 'the': 4844, 'and': 4653, 'of': 42
72, 'I': 3177, '--': 3100, 'a': 3001, "'": 2452, ...})
```

In [33]:
```
da2 = FreqDist(t2)
da2
```

Out[33]:
```
FreqDist({',': 7024, 'the': 3119, '.': 3119, 'to': 2751, 'and': 2724, 'of': 256
2, 'a': 1528, 'in': 1340, 'was': 1330, ';': 1319, ...})
```

In [34]:
```
da3 = FreqDist(t3)
da3
```

Out[34]:
```
FreqDist({',': 9901, 'to': 4050, '.': 4023, 'the': 3860, 'of': 3564, 'and': 334
8, 'her': 2434, 'a': 2025, 'I': 2003, 'in': 1873, ...})
```

G. prints the top 50 word types and their counts.

In [35]:
```
da1.most_common(50)
```

Out[35]:
```
[(',', 12016),
 ('.', 6355),
 ('to', 5125),
 ('the', 4844),
 ('and', 4653),
 ('of', 4272),
 ('I', 3177),
 ('--', 3100),
 ('a', 3001),
 ("''", 2452),
 ('was', 2383),
 ('her', 2360),
 (';', 2353),
 ('not', 2242),
 ('in', 2103),
 ('it', 2103),
 ('be', 1965),
 ('she', 1774),
 ('``', 1735),
 ('that', 1729),
 ('you', 1664),
 ('had', 1605),
 ('as', 1387),
 ('he', 1365),
 ('for', 1320),
 ('have', 1301),
 ('is', 1221),
 ('with', 1185),
 ('very', 1151),
 ('but', 1148),
 ('Mr.', 1091),
 ('his', 1084),
 ('!', 1063),
 ('at', 996),
 ('so', 918),
 ("'s", 866),
 ('Emma', 855),
 ('all', 831),
 ('could', 824),
 ('would', 813),
 ('been', 755),
 ('him', 748),
 ('on', 674),
 ('Mrs.', 668),
 ('any', 651),
 ('?', 621),
 ('my', 619),
 ('no', 616),
 ('Miss', 592),
 ('were', 590)]
```

In [36]:
```
da2.most_common(50)
```

Out[36]:
```
[(',', 7024),
 ('the', 3119),
 ('.', 3119),
 ('to', 2751),
 ('and', 2724),
 ('of', 2562),
 ('a', 1528),
 ('in', 1340),
 ('was', 1330),
 (';', 1319),
 ('had', 1177),
 ('her', 1158),
 ('I', 1123),
 ('not', 968),
 ('be', 949),
 ("''", 912),
 ('it', 857),
 ('that', 853),
 ('she', 819),
 ('as', 787),
 ('he', 736),
 ('for', 695),
 ('``', 652),
 ('with', 643),
 ('his', 625),
 ('have', 583),
 ('but', 553),
 ('you', 548),
 ('at', 519),
 ('all', 517),
 ('Anne', 496),
 ('been', 496),
 ('him', 467),
 ("'s", 464),
 ('could', 444),
 ('were', 426),
 ('very', 425),
 ('which', 415),
 ('by', 409),
 ('is', 393),
 ('on', 386),
 ('would', 351),
 ('so', 338),
 ('She', 327),
 ('they', 323),
 ('!', 318),
 ('no', 309),
 ('Captain', 297),
 ('Mrs', 291),
 ('from', 290)]
```

In [37]:
```
da3.most_common(50)
```

Out[37]:
```
[(',', 9901),
 ('to', 4050),
 ('.', 4023),
 ('the', 3860),
 ('of', 3564),
 ('and', 3348),
 ('her', 2434),
 ('a', 2025),
 ('I', 2003),
 ('in', 1873),
 ('was', 1846),
 ("''", 1807),
 (';', 1572),
 ('it', 1561),
 ('she', 1333),
 ('be', 1304),
 ('not', 1301),
 ('that', 1296),
 ('``', 1277),
 ('for', 1231),
 ('as', 1179),
 ('--', 1178),
 ('you', 1034),
 ('with', 971),
 ('had', 969),
 ('his', 941),
 ('he', 894),
 ('have', 806),
 ('at', 805),
 ('by', 734),
 ('is', 732),
 ('Elinor', 680),
 ('on', 675),
 ("'s", 644),
 ('all', 640),
 ('him', 632),
 ('so', 616),
 ('but', 597),
 ('which', 592),
 ('could', 568),
 ('!', 560),
 ('Marianne', 558),
 ('my', 550),
 ('from', 527),
 ('Mrs.', 523),
 ('would', 507),
 ('very', 492),
 ('no', 488),
 ('their', 463),
 ('them', 460)]
```

EXCERCISE 3

A. imports necessary modules,

B. opens the text files and reads in the content as text strings,

In [38]:
```
with open("jane_austen.txt") as fn:
    nov=fn.read()
print(nov)
```

```
[Emma by Jane Austen 1816]

VOLUME I

CHAPTER I


Emma Woodhouse, handsome, clever, and rich, with a comfortable home
and happy disposition, seemed to unite some of the best blessings
of existence; and had lived nearly twenty-one years in the world
with very little to distress or vex her.

She was the youngest of the two daughters of a most affectionate,
indulgent father; and had, in consequence of her sister's marriage,
been mistress of his house from a very early period.  Her mother
had died too long ago for her to have more than an indistinct
remembrance of her caresses; and her place had been supplied
by an excellent woman as governess, who had fallen little short
of a mother in affection.
```

In [39]:
```
tokenizer = nltk.tokenize.WhitespaceTokenizer()
tok = tokenizer.tokenize(nov)
tok
```

Out[39]:
```
['[Emma',
 'by',
 'Jane',
 'Austen',
 '1816]',
 'VOLUME',
 'I',
 'CHAPTER',
 'I',
 'Emma',
 'Woodhouse,',
 'handsome,',
 'clever,',
 'and',
 'rich,',
 'with',
 'a',
 'comfortable',
 'home',
```

In [40]:
```
b2 = list(nltk.bigrams(tok))
b2fd = nltk.FreqDist(b2)
b2fd
```

Out[40]:
```
FreqDist({('of', 'the'): 1409, ('to', 'be'): 1333, ('in', 'the'): 1086, ('had',
'been'): 668, ('to', 'the'): 645, ('of', 'her'): 601, ('could', 'not'): 573,
('I', 'am'): 569, ('she', 'had'): 548, ('it', 'was'): 546, ...})
```

In [41]:
```
import re
from collections import Counter
```

In [42]:
```
words = re.findall(r'so+ \w+',open('jane_austen.txt').read())
ab = Counter(zip(words))
print(ab)
```

```
Counter({('so much',): 201, ('so very',): 102, ('so well',): 59, ('so man
y',): 54, ('so long',): 50, ('so little',): 44, ('so far',): 40, ('so I',): 2
9, ('so soon',): 23, ('so good',): 20, ('so often',): 16, ('so kind',): 14,
('so great',): 14, ('so it',): 14, ('so entirely',): 11, ('so happy',): 11,
('so you',): 11, ('so near',): 11, ('so to',): 10, ('so anxious',): 10, ('so
easily',): 9, ('so she',): 9, ('so glad',): 9, ('so fond',): 8, ('so ill',):
8, ('so strong',): 8, ('so bad',): 7, ('so as',): 7, ('so lately',): 7, ('so
miserable',): 7, ('so young',): 7, ('so totally',): 6, ('so truly',): 6, ('so
short',): 6, ('so few',): 6, ('so that',): 6, ('so particularly',): 6, ('so f
ull',): 6, ('so large',): 6, ('so extremely',): 6, ('so cheerful',): 6, ('so
pleasantly',): 5, ('so interesting',): 5, ('so completely',): 5, ('so fas
t',): 5, ('so obliging',): 5, ('so lovely',): 5, ('so at',): 5, ('so suddenl
y',): 5, ('so agreeable',): 5, ('so dear',): 4, ('so proper',): 4, ('so bus
y',): 4, ('so forth',): 4, ('so warmly',): 4, ('so charming',): 4, ('so wit
h',): 4, ('so deceived',): 4, ('so odd',): 4, ('so pleased',): 4, ('so deligh
ted',): 4, ('so happened',): 4, ('so thoroughly',): 4, ('so sudden',): 4, ('s
o on',): 4, ('so liberal',): 4, ('so attentive',): 4, ('so he',): 4, ('so sor
ry',): 4, ('so shocked',): 4, ('so wretched',): 4, ('so highly',): 4, ('so de
termined',): 4, ('so does',): 4, ('so unfeeling',): 4, ('so steady',): 4, ('s
```

C. builds the following objects, a_ for Austen:

1. a_toks: word tokens, all in lowercase

In [43]:
```
tokenizer = nltk.tokenize.WhitespaceTokenizer()
a_toks = tokenizer.tokenize(nov.lower())
a_toks
```

Out[43]:
```
['[emma',
 'by',
 'jane',
 'austen',
 '1816]',
 'volume',
 'i',
 'chapter',
 'i',
 'emma',
 'woodhouse,',
 'handsome,',
 'clever,',
 'and',
 'rich,',
 'with',
 'a',
 'comfortable',
 'home',
```

2. a_tokfd: word frequency distribution

In [44]:
```
a_tokfd = FreqDist(a_toks)
a_tokfd
```

Out[44]:
```
FreqDist({'the': 12497, 'to': 11875, 'and': 10444, 'of': 10264, 'a': 6664, 'wa
s': 5363, 'in': 5343, 'i': 5261, 'her': 5238, 'she': 4787, ...})
```

3. a_bigrams: word bigrams, cast as a list

In [45]:
```
a_bigrams = list(nltk.bigrams(a_toks))
a_bigrams
```

Out[45]:
```
[('[emma', 'by'),
 ('by', 'jane'),
 ('jane', 'austen'),
 ('austen', '1816]'),
 ('1816]', 'volume'),
 ('volume', 'i'),
 ('i', 'chapter'),
 ('chapter', 'i'),
 ('i', 'emma'),
 ('emma', 'woodhouse,'),
 ('woodhouse,', 'handsome,'),
 ('handsome,', 'clever,'),
 ('clever,', 'and'),
 ('and', 'rich,'),
 ('rich,', 'with'),
 ('with', 'a'),
 ('a', 'comfortable'),
 ('comfortable', 'home'),
 ('home', 'and'),
```

4. a_bigramfd: bigram frequency distribution

In [46]:
```
a_bigramfd = nltk.FreqDist(a_bigrams)
a_bigramfd
```

Out[46]:
```
FreqDist({('of', 'the'): 1411, ('to', 'be'): 1342, ('in', 'the'): 1115, ('it', 'was'): 826, ('she', 'had'): 715, ('had', 'been'): 669, ('to', 'the'): 650, ('she', 'was'): 648, ('of', 'her'): 601, ('could', 'not'): 576, ...})
```

5. a_bigramcfd: bigram (w1, w2) conditional frequency distribution ("CFD"),where w1 is construed as the condition and w2 the outcome

In [47]:
```
from nltk.probability import ConditionalFreqDist
from nltk.tokenize import word_tokenize
```

In [48]:
```
a_bigramcfd = ConditionalFreqDist()
```

In [49]:
```
for word in a_toks:
    condition = len(word)
    a_bigramcfd[condition][word] += 1
```

In [50]:
```
a_bigramcfd
```

Out[50]:
```
<ConditionalFreqDist with 30 conditions>
```

D. pickles the bigram CFDs (conditional frequency distributions) using the highest binary protocol: name the file as austen_bigramcfd.pkl.

In [51]:
```
with open('austen_bigramcfd.pkl', 'ab') as handle:
    pkl.dump(a_bigramcfd,handle)
```

E. answers the following questions by exploring the objects

1. How many word tokens and types are there? what is its size

In [52]:
```
len(a_toks)
```

Out[52]:
```
360148
```

2. What are the top 20 most frequent words and their counts?. Draw chart using Matplotlib's plot() method.

In [53]:
```
ws=a_tokfd.most_common(20)
n = dict(ws)
n
```

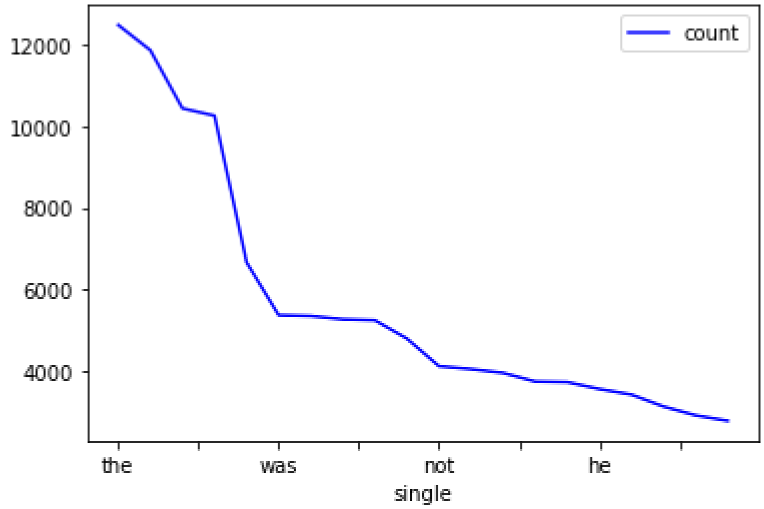
Out[53]:
```
{'the': 12497,
 'to': 11875,
 'and': 10444,
 'of': 10264,
 'a': 6664,
 'was': 5363,
 'in': 5343,
 'i': 5261,
 'her': 5238,
 'she': 4787,
 'not': 4107,
 'be': 4035,
 'it': 3941,
 'had': 3729,
 'that': 3715,
 'he': 3544,
 'as': 3407,
 'for': 3113,
 'you': 2896,
 'his': 2761}
```

In [54]:
```
df = pd.DataFrame(list(n.items()))
df.columns = ['single','count']
df
```

Out[54]:

	single	count
0	the	12497
1	to	11875
2	and	10444
3	of	10264
4	a	6664
5	was	5363
6	in	5343
7	i	5261
8	her	5238
9	she	4787
10	not	4107
11	be	4035
12	it	3941
13	had	3729
14	that	3715
15	he	3544
16	as	3407
17	for	3113
18	you	2896
19	his	2761

In [55]:
```
df.plot(kind='line',x='single',y='count',color='blue')
plt.show()
```



4. What are the top 20 most frequent word bigrams and their counts, omitting bigrams that contain stopwords?

In [56]:
```
v=a_bigramfd.most_common(20)
m = dict(v)
m
```

Out[56]:
```
{('of', 'the'): 1411,
 ('to', 'be'): 1342,
 ('in', 'the'): 1115,
 ('it', 'was'): 826,
 ('she', 'had'): 715,
 ('had', 'been'): 669,
 ('to', 'the'): 650,
 ('she', 'was'): 648,
 ('of', 'her'): 601,
 ('could', 'not'): 576,
 ('i', 'am'): 570,
 ('he', 'had'): 513,
 ('have', 'been'): 495,
 ('of', 'his'): 493,
 ('and', 'the'): 474,
 ('i', 'have'): 474,
 ('he', 'was'): 442,
 ('it', 'is'): 419,
 ('in', 'a'): 408,
 ('for', 'the'): 406}
```

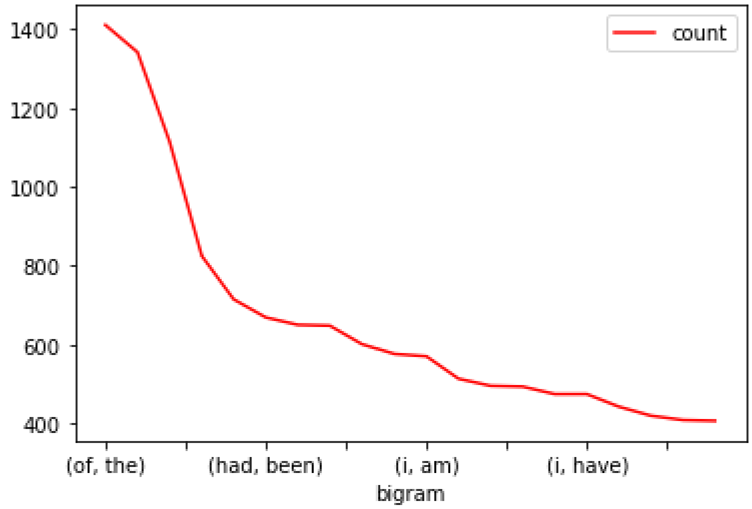
In [57]:
```
df2 = pd.DataFrame(list(m.items()))
df2.columns = ['bigram','count']
df2
```

Out[57]:

	bigram	count
0	(of, the)	1411
1	(to, be)	1342
2	(in, the)	1115
3	(it, was)	826
4	(she, had)	715
5	(had, been)	669
6	(to, the)	650
7	(she, was)	648
8	(of, her)	601
9	(could, not)	576
10	(i, am)	570
11	(he, had)	513
12	(have, been)	495
13	(of, his)	493
14	(and, the)	474
15	(i, have)	474
16	(he, was)	442
17	(it, is)	419
18	(in, a)	408
19	(for, the)	406

5. What are the top 20 most frequent word bigrams and their counts, omitting bigrams that contain stopwords?. Draw chart using Matplotlib's plot() method.

In [58]:
```
df2.plot(kind='line',x='bigram',y='count',color='red')
plt.show()
```



6. How many times does the word 'so' occur? What are their relative frequency against the corpus size (= total # of tokens)?

In [59]:
```
so_count=a_tokfd['so']
print(so_count)

tot=len(a_tokfd)
print(tot)

rel_freq = so_count/tot
rel_freq
```

```
1746
26903
```

Out[59]:
```
0.06489982529829387
```

7. What are the top 20 'so-initial' bigrams (bigrams that have the word “so” as the first word) and their counts?

In [60]:
```
ab.most_common(20)
```

Out[60]:
```
[(('so much',), 201),
 (('so very',), 102),
 (('so well',), 59),
 (('so many',), 54),
 (('so long',), 50),
 (('so little',), 44),
 (('so far',), 40),
 (('so I',), 29),
 (('so soon',), 23),
 (('so good',), 20),
 (('so often',), 16),
 (('so kind',), 14),
 (('so great',), 14),
 (('so it',), 14),
 (('so entirely',), 11),
 (('so happy',), 11),
 (('so you',), 11),
 (('so near',), 11),
 (('so to',), 10),
 (('so anxious',), 10)]
```

8. Given the word 'so' as the current word, what is the probability of getting 'much' as the next word?

In [61]:
```
ab_dict = dict(ab)
ab_dict
```

Out[61]:
```
{('so unperceived',): 1,
 ('so far',): 40,
 ('so obliged',): 2,
 ('so mild',): 1,
 ('so much',): 201,
 ('so to',): 10,
 ('so well',): 59,
 ('so happily',): 3,
 ('so many',): 54,
 ('so long',): 50,
 ('so perfectly',): 3,
 ('so constantly',): 2,
 ('so entirely',): 11,
 ('so comfortably',): 1,
 ('so very',): 102,
 ('so kind',): 14,
 ('so avowed',): 1,
 ('so dear',): 4,
 ('so deservedly',): 1,
```

In [62]:
```
tot_occ=len(ab_dict)
tot_occ
```

Out[62]:
```
584
```

In [63]:
```
for i , j in ab_dict.items():
    if i == ('so much',):
        print(i,j)
        print(j/tot_occ)
```

```
('so much',) 201
0.3441780821917808
```

9. Given the word 'so' as the current word, what is the probability of getting 'will' as the next word?

In [64]:
```
for i , j in ab_dict.items():
    if i == ('so will',):
        print(i,j)
        print(j/tot_occ)
```

```
('so will',) 1
0.0017123287671232876
```

Name:Viviyan Richards W

Roll no:205229133

Lab3. Computing Document Similarity using VSM

EXERCISE-1: Print TFIDF values

In [1]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [2]:
```
import pandas as pd
```

In [3]:
```
docs = ["good movie", "not a good movie", "did not like", "i like it", "good one'
```

In [4]:
```
tfidf = TfidfVectorizer(min_df=2, max_df=0.5, ngram_range=(1, 2))
features = tfidf.fit_transform(docs)
print(features)
```

```
  (0, 0)	0.7071067811865476
  (0, 2)	0.7071067811865476
  (1, 3)	0.5773502691896257
  (1, 0)	0.5773502691896257
  (1, 2)	0.5773502691896257
  (2, 1)	0.7071067811865476
  (2, 3)	0.7071067811865476
  (3, 1)	1.0
```

In [5]:
```
df = pd.DataFrame(
 features.todense(),
 columns=tfidf.get_feature_names())
print(df)
```

```
   good movie      like     movie       not
0    0.707107  0.000000  0.707107  0.000000
1    0.577350  0.000000  0.577350  0.577350
2    0.000000  0.707107  0.000000  0.707107
3    0.000000  1.000000  0.000000  0.000000
4    0.000000  0.000000  0.000000  0.000000
```

EXERCISE-2:

1. Change the values of min_df and ngram_range and observe various outputs

In [6]:
```
tfidf = TfidfVectorizer(min_df=1, max_df=0.6, ngram_range=(1, 2))
features = tfidf.fit_transform(docs)
print(features)
```

```
  (0, 3)	0.6098184563533858
  (0, 8)	0.6098184563533858
  (0, 2)	0.5062044059286201
  (1, 10)	0.5422255279709232
  (1, 9)	0.4374641418373903
  (1, 3)	0.4374641418373903
  (1, 8)	0.4374641418373903
  (1, 2)	0.36313475547801904
  (2, 11)	0.4821401170833009
  (2, 1)	0.4821401170833009
  (2, 6)	0.3889876106617681
  (2, 0)	0.4821401170833009
  (2, 9)	0.3889876106617681
  (3, 7)	0.6141889663426562
  (3, 5)	0.6141889663426562
  (3, 6)	0.49552379079705033
  (4, 4)	0.6390704413963749
  (4, 12)	0.6390704413963749
  (4, 2)	0.42799292268317357
```

In [7]:
```
df = pd.DataFrame(
 features.todense(),
 columns=tfidf.get_feature_names())
print(df)
```

```
       did  did not      good  good movie  good one        it      like  \
0  0.00000  0.00000  0.506204    0.609818   0.00000  0.000000  0.000000   
1  0.00000  0.00000  0.363135    0.437464   0.00000  0.000000  0.000000   
2  0.48214  0.48214  0.000000    0.000000   0.00000  0.000000  0.388988   
3  0.00000  0.00000  0.000000    0.000000   0.00000  0.614189  0.495524   
4  0.00000  0.00000  0.427993    0.000000   0.63907  0.000000  0.000000   

    like it     movie       not  not good  not like      one  
0  0.000000  0.609818  0.000000  0.000000   0.00000  0.00000  
1  0.000000  0.437464  0.437464  0.542226   0.00000  0.00000  
2  0.000000  0.000000  0.388988  0.000000   0.48214  0.00000  
3  0.614189  0.000000  0.000000  0.000000   0.00000  0.00000  
4  0.000000  0.000000  0.000000  0.000000   0.00000  0.63907  
```

EXERCISE-3: Compute Cosine Similarity between 2 Documents

In [8]:
```
from sklearn.metrics.pairwise import linear_kernel
```

In [9]:
```
doc1 = features[0:1]
doc2 = features[1:2]
score = linear_kernel(doc1, doc2)
print(score)
```

```
[[0.71736783]]
```

In [10]:
```
scores = linear_kernel(doc1, features)
print(scores)
```

```
[[1.         0.71736783 0.         0.         0.2166519 ]]
```

In [11]:
```
query = "I like this good movie"
qfeature = tfidf.transform([query])
scor = linear_kernel(doc1, features)
print(scor)
```

```
[[1.         0.71736783 0.         0.         0.2166519 ]]
```

EXERCISE-4: Find Top-N similar documents

Question-1. Consider the following documents and compute TFIDF values

In [12]:
```
docs=["the house had a tiny little mouse",
"the cat saw the mouse",
"the mouse ran away from the house",
"the cat finally ate the mouse",
"the end of the mouse story"
]
```

Question-2. Compute cosine similarity between 3rd document (“the mouse ran away from the house”) with all other documents. Which is the most similar document?

In [13]:
```
tfidf = TfidfVectorizer(min_df=2, max_df=0.5, ngram_range=(1, 2))
features = tfidf.fit_transform(docs)
print(features)
```

```
  (0, 3)	0.7071067811865476
  (0, 1)	0.7071067811865476
  (1, 2)	0.7071067811865476
  (1, 0)	0.7071067811865476
  (2, 3)	0.7071067811865476
  (2, 1)	0.7071067811865476
  (3, 2)	0.7071067811865476
  (3, 0)	0.7071067811865476
```

In [14]:
```
doc1=features[0:3]
s=linear_kernel(doc1,features)
print(s)
```

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

In [15]:
```
scores2 = linear_kernel(doc1, features)
print(scores2)
```

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

Name:Viviyan Richards

Roll no:205229133

EXERCISE-1

1. Import dependencies

In [1]:
```
import gensim
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from nltk.tokenize import word_tokenize
from sklearn import utils
```

In [2]:
```
data = ["I love machine learning. Its awesome.",
"I love coding in python",
"I love building chatbots",
"they chat amagingly well"]
```

In [3]:
```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Angelan\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[3]: True

In [4]:
```
tagged_data = [TaggedDocument(words=word_tokenize(d.lower()),
tags=[str(i)]) for i, d in enumerate(data)]
```

In [5]:
```
vec_size = 20
alpha = 0.025
```

In [6]:
```
model = Doc2Vec(vector_size=vec_size,
alpha=alpha,
min_alpha=0.00025,
min_count=1,
dm =1)
# build vocabulary
model.build_vocab(tagged_data)
# shuffle data
tagged_data = utils.shuffle(tagged_data)
# train Doc2Vec model
model.train(tagged_data,
total_examples=model.corpus_count,
epochs=30)
model.save("d2v.model")
print("Model Saved")
```

```
Model Saved
```

In [7]:
```
from gensim.models.doc2vec import Doc2Vec
model= Doc2Vec.load("d2v.model")
#to find the vector of a document which is not in training data
test_data = word_tokenize("I love chatbots".lower())
v1 = model.infer_vector(test_data)
print("V1_infer", v1)
```

```
V1_infer [ 0.0032945   0.0009504   0.01332451  0.01152915  0.01895528  0.023096
65
 -0.00325777 -0.00802977  0.0097452  -0.023578    0.01137165  0.01260952
 -0.00895888  0.00068234 -0.00607778 -0.00854787  0.00213298  0.01996933
  0.00269892  0.00989255]
```

In [8]:
```
similar_doc = model.docvecs.most_similar('1')
print(similar_doc)
```

```
[('0', 0.1854361891746521), ('2', -0.03567519038915634), ('3', -0.0862233042716
98)]
```

In [9]:
```
print(model.docvecs['1'])
```

```
[ 0.00233202 -0.0020763  -0.01821837 -0.02302309  0.00686011  0.01970871
  0.02488494 -0.01114094  0.02446651  0.00846515 -0.00418958 -0.00347237
  0.01749527 -0.02282372 -0.00218709 -0.01023882 -0.01316169  0.02423306
  0.01739944 -0.01872601]
```

In [10]:
```
docs=["the house had a tiny little mouse",
"the cat saw the mouse",
"the mouse ran away from the house",
"the cat finally ate the mouse",
"the end of the mouse story"
]
```

In [11]:
```
tagged_data = [TaggedDocument(words=word_tokenize(d.lower()),
tags=[str(i)]) for i, d in enumerate(docs)]
```

In [16]:
```
vec_size = 20
alpha = 0.025
# create model
model = Doc2Vec(vector_size=vec_size, alpha=alpha, min_alpha=0.00025,min_count=1,
```

In [17]:
```
model.build_vocab(tagged_data)
```

In [18]:
```
tagged_data = utils.shuffle(tagged_data)
```

In [19]:
```
model.train(tagged_data,total_examples=model.corpus_count,epochs=30)
model.save("d2v.model")
print("Model Saved")
```

```
Model Saved
```

In [20]:
```
from gensim.models.doc2vec import Doc2Vec
model= Doc2Vec.load("d2v.model")
```

In [21]:
```
test_data = word_tokenize("cat stayed in the house".lower())
v1 = model.infer_vector(test_data)
print("V1_infer", v1)
```

```
V1_infer [ 0.00522686 -0.02354816 -0.00739392  0.01496425 -0.02397058  0.019779
99
  0.02080073  0.01991114 -0.00643659  0.00969159  0.01156812  0.00580886
 -0.00234774 -0.00706865  0.02135056  0.0247726  -0.00312962  0.02308429
 -0.01610882 -0.01583623]
```

In [22]:
```
similar_doc = model.docvecs.most_similar('2')
print(similar_doc)
```

```
[('0', 0.07634814828634262), ('4', -0.017351791262626648), ('1', -0.02717830240
726471), ('3', -0.48652878403663635)]
```

In [ ]:

In [1]:
```
from zipfile import ZipFile
import glob
import pandas as pd
import nltk
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
from nltk.corpus import stopwords
import warnings
warnings.filterwarnings('ignore')
```

EXERCISE-1

The file movie.zip contains 20 files about various movies. For each of the files in movies.zip, you will have to do the following:

In [2]:
```
file_name = "movies.zip"            # opening the zip file in READ mode
with ZipFile(file_name, 'r') as zip:
    zip.printdir()                  # printing all the contents of the zip file
```

```
File Name                                             Modified             Size
movies/Three Colors Red.txt                    2021-05-04 04:18:04         2892
movies/The Godfather.txt                       2021-05-04 04:18:02         4293
movies/Some Like It Hot.txt                    2021-05-04 04:18:02         7489
movies/Ran.txt                                 2021-05-04 04:18:02         2207
movies/Psycho.txt                              2021-05-04 04:18:00         3727
movies/Pan_s Labyrinth.txt                     2021-05-04 04:18:00         4431
movies/My Left Foot.txt                        2021-05-04 04:17:58         1115
movies/Moonlight.txt                           2021-05-04 04:17:58         2323
movies/Manchester by the Sea.txt               2021-05-04 04:17:58         3674
movies/Hoop Dreams.txt                         2021-05-04 04:17:58         7909
movies/Citizen Kane.txt                        2021-05-04 04:17:56         1483
movies/Gone with the Wind.txt                  2021-05-04 04:17:56         1318
movies/Casablanca.txt                          2021-05-04 04:17:54         1896
movies/American Graffiti.txt                   2021-05-04 04:17:54         3417
movies/4 Months, 3 Weeks and 2 Days.txt        2021-05-04 04:17:52         1151
movies/All About Eve.txt                       2021-05-04 04:17:52         1346
movies/12 Angry Men.txt                        2021-05-04 04:17:52         1007
movies/12 Years a Slave.txt                    2021-05-04 04:17:52         6451
movies/Singin_ in the Rain.txt                 2021-05-04 04:18:02          782
```

In [3]:
```
files = [file for file in glob.glob("movies/*")]
files
```

Out[3]:
```
['movies\\12 Angry Men.txt',
 'movies\\12 Years a Slave.txt',
 'movies\\4 Months, 3 Weeks and 2 Days.txt',
 'movies\\All About Eve.txt',
 'movies\\American Graffiti.txt',
 'movies\\Boyhood.txt',
 'movies\\Casablanca.txt',
 'movies\\Citizen Kane.txt',
 'movies\\Gone with the Wind.txt',
 'movies\\Hoop Dreams.txt',
 'movies\\Manchester by the Sea.txt',
 'movies\\Moonlight.txt',
 'movies\\My Left Foot.txt',
 "movies\\Pan's Labyrinth.txt",
 'movies\\Psycho.txt',
 'movies\\Ran.txt',
 "movies\\Singin' in the Rain.txt",
 'movies\\Some Like It Hot.txt',
 'movies\\The Godfather.txt',
 'movies\\Three Colors Red.txt']
```

In [4]:
```
nltk.download('punkt')
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
```

```
[nltk_data] Error loading punkt: <urlopen error [Errno 11001]
[nltk_data]     getaddrinfo failed>
[nltk_data] Error loading stopwords: <urlopen error [Errno 11001]
[nltk_data]     getaddrinfo failed>
```

In [5]:
```
tokenizer = nltk.tokenize.WhitespaceTokenizer()
from nltk.stem import PorterStemmer
ps =PorterStemmer()
from nltk.stem import LancasterStemmer
ls = LancasterStemmer()
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
```

In [6]:
```
for file in files:
    with open(file,'r',encoding='cp1252') as f:
        contents = f.readlines()
        print(contents)
        print("**********************************************************")
        print("                                                          ")
```

```
["Lumet's origins as a director of teledrama may well be obvious here in his
first film, but there is no denying the suitability of his style - sweaty clo
se-ups, gritty monochrome 'realism', one-set claustrophobia - to his subject.
Scripted by Reginald Rose from his own teleplay, the story is pretty contrive
d - during a murder trial, one man's doubts about the accused's guilt gradual
ly overcome the rather less-than-democratic prejudices of the other eleven me
mbers of the jury - but the treatment is tense, lucid, and admirably economic
al. Fonda, though typecast as the bastion of liberalism, gives a nicely under
played performance, while Cobb, Marshall and Begley in particular are highly
effective in support. But what really transforms the piece from a rather talk
y demonstration that a man is innocent until proven guilty, is the consistent
ly taut, sweltering atmosphere, created largely by Boris Kaufman's excellent
camerawork. The result, however devoid of action, is a strangely realistic th
riller."]
**********************************************************


['There are movies to which the critical response lags far behind the emotion
al one. Two days after seeing 12 Years a Slave, British director Steve McQuee
n’s adaptation of the 1853 memoir of a free black man kidnapped into slavery,
```

A. How many sentences in each file?

B. How many tokens in each file?

C. How many tokens excluding stop words in each file?

In [7]:
```
files = [file for file in glob.glob("movies/*")]
for file in files:
    with open(file, 'r',encoding='cp1252') as f:
        contents = f.readlines()
        for row in contents:
            sent_text = nltk.sent_tokenize(row)
        print("sentence tokenize  ",len(sent_text))
        for row1 in contents:
            words =nltk.word_tokenize(row1)
        print("word tokenize  ",len(words))
        filtered_sentence = [w for w in words if not w in stop_words]
        print("stopwords  ",len(filtered_sentence))
        print("***********")
```

```
sentence tokenize   5
word tokenize   181
stopwords   122
***********
sentence tokenize   2
word tokenize   119
stopwords   68
***********
sentence tokenize   1
word tokenize   20
stopwords   11
***********
sentence tokenize   7
word tokenize   276
stopwords   178
***********
sentence tokenize   1
word tokenize   9
stopwords   7
***********
```

D. How many unique stems (ie., stemming) in each file? (Use PorterStemmer)

In [8]:
```
def port_stemSentence(sentence):
    tok = tokenizer.tokenize(sentence)
    filtered_sentence = [w for w in tok if not w in stop_words]
    stem_sentence=[]
    for word in filtered_sentence:
        stem_sentence.append(ps.stem(word))
    return len(stem_sentence)
```

In [9]:
```
for file in files:
    with open(file, 'r',encoding='cp1252') as f:
        contents = f.readline()
        print("porter_stemming  ")
        print(port_stemSentence(contents))
    print("******************")
```

```
porter_stemming
96
******************
porter_stemming
83
******************
porter_stemming
20
******************
porter_stemming
138
******************
porter_stemming
63
******************
porter_stemming
64
******************
porter_stemming
```

E. How many unique stems (ie., stemming) in each file? (Use LancasterStemmer)

In [10]:
```
def lan_stemSentence(sentence):
    tok = tokenizer.tokenize(sentence)
    filtered_sentence = [w for w in tok if not w in stop_words]
    stem_sentence=[]
    for word in filtered_sentence:
        stem_sentence.append(ls.stem(word))
    return len(stem_sentence)
```

In [11]:
```
for file in files:
    with open(file, 'r',encoding='cp1252') as f:
        contents = f.readline()
        print("lancaster_stemming  ")
        print(lan_stemSentence(contents))
    print("******************")
```

```
lancaster_stemming
96
******************
lancaster_stemming
83
******************
lancaster_stemming
20
******************
lancaster_stemming
138
******************
lancaster_stemming
63
******************
lancaster_stemming
64
******************
lancaster_stemming
```

F. How many unique words (ie., lemmatization) in each file? (Use WordNetLemmatizer)

In [12]:
```
def lemmSentence(sentence):
    tok = tokenizer.tokenize(sentence)
    filtered_sentence = [w for w in tok if not w in stop_words]
    lemm_sentence=[]
    for word in filtered_sentence:
        lemm_sentence.append(lemmatizer.lemmatize(word))
    return len(lemm_sentence)
```

In [13]:
```python
for file in files:
    with open(file, 'r',encoding='cp1252') as f:
        contents = f.readline()
        print("lemmatization ")
        print(lemmSentence(contents))
    print("*******************")
```

```
lemmatization
96
*******************
lemmatization
83
*******************
lemmatization
20
*******************
lemmatization
138
*******************
lemmatization
63
*******************
lemmatization
64
*******************
lemmatization
```

EXERCISE-2

In this exercise, you will build your Term-Document Matrix for this movie collection of 20 movies. In order to improve the similarity search experience, you will use only lemmatized terms for creating the matrix.

Step-1 For each movie:

1. Tokenize terms and build list of tokens
2. Find lemmatized words from the tokens

In [14]:
```
tok = []
for file in files:
    with open(file,'r',encoding='cp1252') as f:
        contents = f.read()
        let=tokenizer.tokenize(contents)
        tok.append(let)
tok
```

Out[14]:
```
[["Lumet's",
  'origins',
  'as',
  'a',
  'director',
  'of',
  'teledrama',
  'may',
  'well',
  'be',
  'obvious',
  'here',
  'in',
  'his',
  'first',
  'film,',
  'but',
  'there',
  'is',
```

In [15]:
```
tok_lem =[]
for i in tok:
    for j in i:
        to_lem = lemmatizer.lemmatize(j)
        tok_lem.append(to_lem)
tok_lem
```

Out[15]:
```
["Lumet's",
 'origin',
 'a',
 'a',
 'director',
 'of',
 'teledrama',
 'may',
 'well',
 'be',
 'obvious',
 'here',
 'in',
 'his',
 'first',
 'film,',
 'but',
 'there',
 'is',
```

Step-2

Build Term-Document matrix using TfIdfVectorizer

In [16]:
```
for file in files:
    with open(file,'r',encoding='cp1252') as f:
        contents = f.read()
        tok = tokenizer.tokenize(contents)
        filtered_sentence = [w for w in tok if not w in stop_words]
        tfidf = TfidfVectorizer(min_df=2,max_df=0.5,ngram_range=(1,2))
        features = tfidf.fit_transform(filtered_sentence)
        df = pd.DataFrame(features.todense(),columns=tfidf.get_feature_names())
        print(df)
        print("*****************")
```

```
    man  one  rather
0   0.0  0.0     0.0
1   0.0  0.0     0.0
2   0.0  0.0     0.0
3   0.0  0.0     0.0
4   0.0  0.0     0.0
..  ...  ...     ...
91  0.0  0.0     0.0
92  0.0  0.0     0.0
93  0.0  0.0     0.0
94  0.0  0.0     0.0
95  0.0  0.0     0.0

[96 rows x 3 columns]
*****************
     12  all  almost  and  beautiful  black  but  children  comes  cotton  \
0   0.0  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  0.0        0.0    0.0  0.0       0.0    0.0     0.0
2   0.0  0.0     0.0  0.0        0.0    0.0  0.0       0.0    0.0     0.0
```

Step-3

Take vectors of any two movies and compute cosine similarity

In [17]:
```
with open(files[5],'r',encoding='cp1252')as f:
    contents = f.read()
    tok = tokenizer.tokenize(contents)
    filtered_sentence = [w for w in tok if not w in stop_words]
    tfidf = TfidfVectorizer(min_df=2,max_df=0.5,ngram_range=(1,2))
    movie1 = tfidf.fit_transform(filtered_sentence)
    print(movie1)
```

```
  (1, 10)	1.0
  (5, 2)	1.0
  (12, 13)	1.0
  (15, 5)	1.0
  (18, 10)	1.0
  (31, 20)	1.0
  (35, 12)	1.0
  (37, 3)	1.0
  (38, 9)	1.0
  (45, 10)	1.0
  (46, 11)	1.0
  (48, 19)	1.0
  (49, 16)	1.0
  (53, 8)	1.0
  (54, 4)	1.0
  (56, 19)	1.0
  (62, 20)	1.0
  (65, 12)	1.0
  (69, 7)	1.0
```

In [18]:
```
with open(files[10],'r',encoding='cp1252')as f:
    contents = f.read()
    tok = tokenizer.tokenize(contents)
    filtered_sentence = [w for w in tok if not w in stop_words]
    tfidf = TfidfVectorizer(min_df=2,max_df=0.5,ngram_range=(1,2))
    movie2 = tfidf.fit_transform(filtered_sentence)
    print(movie2)
```

```
  (0, 15)	1.0
  (1, 27)	1.0
  (2, 34)	1.0
  (3, 6)	1.0
  (4, 8)	1.0
  (7, 26)	1.0
  (11, 22)	1.0
  (13, 19)	1.0
  (15, 20)	1.0
  (17, 0)	1.0
  (29, 11)	1.0
  (34, 16)	1.0
  (46, 35)	1.0
  (52, 43)	1.0
  (53, 20)	1.0
  (62, 11)	1.0
  (66, 20)	1.0
  (67, 10)	1.0
  (71, 14)	1.0
```

In [19]:
```
doc1 = movie1[0:10]
doc2 = movie1[:]
score = linear_kernel(doc1,doc2)
print(score)
```

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

In [1]:
```
import pandas as pd
from nltk.corpus import stopwords
```

1. Open “SMSSpamCollection” file and load into DataFrame. It contains two columns “label” and “text" ¶

In [2]:
```
df = pd.read_csv("SMSSpamCollection.csv")
```

In [3]:
```
sms=df.drop(['Unnamed: 2','Unnamed: 3','Unnamed: 4'],axis=1)
sms
```

Out[3]:

	label	text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will �_ b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

5572 rows × 2 columns

2. How many sms messages are there?

In [4]:
```
len(sms)
```

Out[4]: 5572

3. How many “ham” and “spam” messages?. You need to groupby() label column.

In [5]:
```
a=sms.groupby('label').count()
a
```

Out[5]:

	text
label	
ham	4825
spam	747

4. Split the dataset into training set and test set (Use 20% of data for testing).

In [6]:
```
X = sms.text
y = sms.label
```

In [7]:
```
from sklearn.model_selection import train_test_split
```

In [8]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,train_size=0.8,test_size=0.2
```

5. Create a function that will remove all punctuation characters and stop words, as below

In [9]:
```
def process_text(msg):
    punctuations = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''
    nopunc =[char for char in msg if char not in punctuations]
    nopunc=''.join(nopunc)
    return [word for word in nopunc.split()
    if word.lower() not in stopwords.words('english')]
```

6. Create TfIdfVectorizer as below and perform vectorization on X_train, using fit_perform() method.

In [10]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [11]:
```
tf2=TfidfVectorizer(use_idf=True,analyzer=process_text,ngram_range=(1,3),min_df =
tf2
```

Out[11]:
```
TfidfVectorizer(analyzer=<function process_text at 0x000001D510FDA820>,
                ngram_range=(1, 3), stop_words='english')
```

In [12]:
```
m2=tf2.fit_transform(X_train)
my2=tf2.transform(X_test)
```

In [13]:
```
m2.shape
```
Out[13]:
```
(4457, 9960)
```

In [14]:
```
my2.shape
```
Out[14]:
```
(1115, 9960)
```

7. Create MultinomialNB model and perform training on X_train and y_train using fit() method

In [15]:
```
x_train,x_test,Y_train,Y_test = train_test_split(X,y,train_size=0.8,test_size=0.2
```

In [16]:
```
from sklearn.naive_bayes import MultinomialNB
```

In [17]:
```
clf = MultinomialNB()
```

In [18]:
```
clf.fit(m2,y_train)
```
Out[18]:
```
MultinomialNB()
```

8. Predict labels on the test set, using predict() method

In [19]:
```
y_pred = clf.predict(my2)
y_pred
```
Out[19]:
```
array(['ham', 'ham', 'ham', ..., 'ham', 'ham', 'ham'], dtype='<U4')
```

9. Print confusion_matrix and classification_report

In [20]:
```
from sklearn.metrics import confusion_matrix
```

In [21]:
```
confusion_matrix(y_test,y_pred)
```
Out[21]:
```
array([[952,   0],
       [ 47, 116]], dtype=int64)
```

In [22]:
```
from sklearn.metrics import classification_report
```

In [32]:
```
target_names = ['class 0', 'class 1']
print(classification_report(y_test,y_pred,target_names=target_names))
```

```
              precision    recall  f1-score   support

     class 0       0.95      1.00      0.98       952
     class 1       1.00      0.71      0.83       163

    accuracy                           0.96      1115
   macro avg       0.98      0.86      0.90      1115
weighted avg       0.96      0.96      0.95      1115
```

10. Modify ngram_range=(1,2) and perform Steps 7 to 9.

In [24]:
```
tf3=TfidfVectorizer(use_idf=True,analyzer=process_text,ngram_range=(1,2),min_df =
tf3
```

Out[24]:
```
TfidfVectorizer(analyzer=<function process_text at 0x000001D510FDA820>,
                ngram_range=(1, 2), stop_words='english')
```

In [25]:
```
m3=tf3.fit_transform(X_train)
my3=tf3.transform(X_test)
```

In [26]:
```
m3.shape
```

Out[26]:
```
(4457, 9960)
```

In [27]:
```
my3.shape
```

Out[27]:
```
(1115, 9960)
```

In [28]:
```
clf.fit(m3,y_train)
```

Out[28]:
```
MultinomialNB()
```

In [29]:
```
y_pred3 = clf.predict(my3)
y_pred3
```

Out[29]:
```
array(['ham', 'ham', 'ham', ..., 'ham', 'ham', 'ham'], dtype='<U4')
```

In [30]:
```
confusion_matrix(y_test,y_pred3)
```

Out[30]:
```
array([[952,   0],
       [ 47, 116]], dtype=int64)
```

In [31]:
```
target_names = ['class 0', 'class 1']
print(classification_report(y_test,y_pred3,target_names=target_names))
```

```
              precision    recall  f1-score   support

     class 0       0.95      1.00      0.98       952
     class 1       1.00      0.71      0.83       163

    accuracy                           0.96      1115
   macro avg       0.98      0.86      0.90      1115
weighted avg       0.96      0.96      0.95      1115
```

In [1]:
```
import pandas as pd
```

EXERCISE-1

1. Open the file, 'rotten_tomato_train.tsv' and read into a DataFrame

In [2]:
```
rotten_tomato_train = pd.read_csv('rotten_tomato_train.tsv', sep='\t')
```

In [3]:
```
rotten_tomato_train.head()
```

Out[3]:

	PhraseId	SentenceId	Phrase	Sentiment
0	1	1	A series of escapades demonstrating the adage ...	1
1	2	1	A series of escapades demonstrating the adage ...	2
2	3	1	A series	2
3	4	1	A	2
4	5	1	series	2

2. Print the basic statistics such as head, shape, describe, and columns

In [4]:
```
rotten_tomato_train.tail()
```

Out[4]:

	PhraseId	SentenceId	Phrase	Sentiment
156055	156056	8544	Hearst 's	2
156056	156057	8544	forced avuncular chortles	1
156057	156058	8544	avuncular chortles	3
156058	156059	8544	avuncular	2
156059	156060	8544	chortles	2

In [5]:
```
rotten_tomato_train.shape
```

Out[5]:
```
(156060, 4)
```

In [6]:
```
rotten_tomato_train.describe
```

Out[6]:
```
<bound method NDFrame.describe of         PhraseId  SentenceId  \
0              1           1
1              2           1
2              3           1
3              4           1
4              5           1
...          ...         ...
156055    156056        8544
156056    156057        8544
156057    156058        8544
156058    156059        8544
156059    156060        8544

                                                   Phrase  Sentiment
0       A series of escapades demonstrating the adage ...          1
1       A series of escapades demonstrating the adage ...          2
2                                                A series          2
3                                                       A          2
4                                                  series          2
...                                                   ...        ...
156055                                          Hearst 's          2
156056                          forced avuncular chortles          1
156057                                 avuncular chortles          3
156058                                          avuncular          2
156059                                           chortles          2

[156060 rows x 4 columns]>
```

In [7]:
```
rotten_tomato_train.columns
```

Out[7]:
```
Index(['PhraseId', 'SentenceId', 'Phrase', 'Sentiment'], dtype='object')
```

3. How many reviews exist for each sentiment?

In [8]:
```
review=rotten_tomato_train.groupby('Sentiment').count()
review.Phrase
```

Out[8]:
```
Sentiment
0     7072
1    27273
2    79582
3    32927
4     9206
Name: Phrase, dtype: int64
```

EXERCISE-2

1. Extract 200 reviews for each sentiment, store them into a new dataframe and create a smaller dataset. Save this dataframe in a new

file, say, “small_rotten_train.csv”.

In [9]:
```
a=rotten_tomato_train.loc[rotten_tomato_train.Sentiment == 0]
b=rotten_tomato_train.loc[rotten_tomato_train.Sentiment == 1]
c=rotten_tomato_train.loc[rotten_tomato_train.Sentiment == 2]
d=rotten_tomato_train.loc[rotten_tomato_train.Sentiment == 3]
e=rotten_tomato_train.loc[rotten_tomato_train.Sentiment == 4]
```

In [10]:
```
small_rotten_train=pd.concat([a[:200],b[:200],c[:200],d[:200],e[:200]])
```

EXERCISE-3

1. Open the file, “small_rotten_train.csv”.

In [11]:
```
small_rotten_train
```

Out[11]:

	PhraseId	SentenceId	Phrase	Sentiment
101	102	3	would have a hard time sitting through this one	0
103	104	3	have a hard time sitting through this one	0
157	158	5	Aggressive self-glorification and a manipulati...	0
159	160	5	self-glorification and a manipulative whitewash	0
201	202	7	Trouble Every Day is a plodding mess .	0
...	...	...	...	...
3744	3745	142	amazing slapstick	4
3745	3746	142	amazing	4
3847	3848	147	When cowering and begging at the feet a scruff...	4
3866	3867	147	gives her best performance since Abel Ferrara ...	4
3993	3994	151	Spielberg 's realization of a near-future Amer...	4

1000 rows × 4 columns

2. The review text are stored in “Phrase” column. Extract that into a separate DataFrame, say “X”.

In [12]:
```
X = small_rotten_train.Phrase
```

3. The “sentiment” column is your target, say “y”.

In [13]:
```
y = small_rotten_train.Sentiment
```

4. Perform pre-processing: convert into lower case, remove stop words and lemmatize. The following function will help.

In [14]:
```
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
```

```
[nltk_data] Downloading package stopwords to C:\Users\Arzoo
[nltk_data]     Sah\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [15]:
```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
```

In [16]:
```
def clean_review(review):
    tokens = review.lower().split()
    filtered_tokens = [lemmatizer.lemmatize(w) for w in tokens if w not in stop_w
    return " ".join(filtered_tokens)
```

5. Apply the above function to X

In [17]:
```
temp=X.tolist()
fax=[]
for i in temp:
    fax.append(clean_review(i))
n_X=pd.Series(fax)
```

6. Split X and y for training and testing (Use 20% for testing)

In [18]:
```
from sklearn.model_selection import train_test_split
```

In [19]:
```
X_train,X_test,y_train,y_test = train_test_split(n_X,y,train_size=0.8,test_size=0
```

7. Create TfidfVectorizer as below and perfrom vectorization on X_train using fit_perform() method.

In [20]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [21]:
```
tf=TfidfVectorizer(min_df=3, max_features=None,ngram_range=(1, 2), use_idf=1)
tf
```

Out[21]:
```
TfidfVectorizer(min_df=3, ngram_range=(1, 2), use_idf=1)
```

In [22]:
```
m=tf.fit_transform(X_train)
m.shape
```

Out[22]:
```
(800, 874)
```

8. Create MultinomialNB model and perform training using X_train_lemmartized and y_train.

In [23]:
```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
```

In [24]:
```
X_train_dtm = cv.fit_transform(X_train)
X_test_dtm = cv.transform(X_test)
```

In [25]:
```
from sklearn.naive_bayes import MultinomialNB
```

In [26]:
```
clf = MultinomialNB()
```

In [27]:
```
clf.fit(X_train_dtm,y_train)
```

Out[27]:
```
MultinomialNB()
```

9. Perform validation on X_test lemmatized and predict output

In [28]:
```
y_real_pred = clf.predict(X_test_dtm)
y_real_pred
```

Out[28]:
```
array([2, 4, 4, 0, 0, 4, 2, 2, 3, 3, 1, 3, 0, 3, 2, 4, 1, 1, 1, 2, 3, 4,
       3, 4, 3, 3, 4, 3, 1, 3, 1, 3, 4, 0, 3, 2, 0, 2, 1, 4, 1, 0, 3, 1,
       1, 1, 3, 3, 1, 0, 3, 1, 3, 3, 0, 0, 2, 1, 2, 2, 2, 4, 3, 3, 3, 3,
       0, 2, 1, 1, 4, 2, 3, 2, 3, 2, 4, 3, 1, 2, 4, 4, 3, 3, 1, 3, 4, 1,
       2, 3, 1, 3, 3, 3, 2, 3, 4, 4, 3, 2, 2, 1, 3, 3, 1, 2, 3, 2, 3, 1,
       2, 1, 3, 4, 2, 0, 4, 0, 4, 0, 4, 1, 0, 3, 1, 3, 1, 0, 3, 3, 0, 3,
       1, 0, 2, 0, 4, 3, 0, 3, 0, 2, 0, 3, 2, 0, 4, 1, 4, 2, 0, 1, 4, 3,
       2, 2, 2, 2, 0, 0, 3, 3, 2, 0, 3, 1, 4, 2, 1, 1, 0, 3, 1, 4, 0, 4,
       4, 2, 1, 2, 0, 0, 2, 1, 4, 3, 1, 4, 1, 3, 4, 1, 0, 3, 0, 1, 4, 3,
       2, 3], dtype=int64)
```

10. Print classification_report and accuracy score.

In [29]:
```
from sklearn.metrics import classification_report
```

In [30]:
```
print(classification_report(y_test,y_real_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.73      0.78        37
           1       0.65      0.59      0.62        44
           2       0.66      0.54      0.60        46
           3       0.44      0.76      0.56        33
           4       0.73      0.60      0.66        40

    accuracy                           0.64       200
   macro avg       0.66      0.64      0.64       200
weighted avg       0.67      0.64      0.64       200
```

In [31]:
```
from sklearn.metrics import accuracy_score
```

In [32]:
```
accuracy_score(y_test,y_real_pred)
```

Out[32]:
```
0.635
```

EXERCISE-4

In [33]:
```
rotten_tomato_test = pd.read_csv('rotten_tomato_test.tsv', sep='\t')
```

In [34]:
```
rotten_tomato_test.shape
```

Out[34]:
```
(66292, 3)
```

In [35]:
```
X_ = rotten_tomato_test.Phrase
```

In [36]:
```
t_temp=X_.tolist()
t_fax=[]
for i in t_temp:
    t_fax.append(clean_review(i))
nt_X=pd.Series(t_fax)
```

In [37]:
```
nt_X
```

Out[37]:
```
0        intermittently pleasing mostly routine effort .
1          intermittently pleasing mostly routine effort
2
3          intermittently pleasing mostly routine effort
4                 intermittently pleasing mostly routine
                                ...
66287                long-winded , predictable scenario .
66288                  long-winded , predictable scenario
66289                                       long-winded ,
66290                                         long-winded
66291                                predictable scenario
Length: 66292, dtype: object
```

In [38]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [39]:
```
tf2=TfidfVectorizer(use_idf=True,ngram_range=(1,3),min_df = 1)
tf2
```

Out[39]:
```
TfidfVectorizer(ngram_range=(1, 3))
```

In [41]:
```
my2=tf2.fit_transform(nt_X)
my2
```

Out[41]:
```
<66292x61283 sparse matrix of type '<class 'numpy.float64'>'
	with 571899 stored elements in Compressed Sparse Row format>
```

In [ ]:

205229133

In [1]:

```
from zipfile import ZipFile
import glob
import nltk
import pandas as pd
from nltk import *
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\ELCOT\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

Out[1]:

```
True
```

1. Open any movie file from your movies sub directory

In [2]:

```
files = [file for file in glob.glob("movies/*")]
#for file in files[:1]:
with open(files[-3],'r',encoding='cp1252') as f:
    cont = f.read()
    print(cont)
```

```
What a work of art and nature is Marilyn Monroe. She hasn't aged into an ico
n, some citizen of the past, but still seems to be inventing herself as we w
atch her. She has the gift of appearing to hit on her lines of dialogue by h
appy inspiration, and there are passages in Billy Wilder's "Some Like It Ho
t" where she and Tony Curtis exchange one-liners like hot potatoes.

Poured into a dress that offers her breasts like jolly treats for needy boy
s, she seems totally oblivious to sex while at the same time melting men int
o helpless desire. "Look at that!" Jack Lemmon tells Curtis as he watches he
r adoringly. "Look how she moves. Like Jell-O on springs. She must have some
sort of built-in motor. I tell you, it's a whole different sex."

Wilder's 1959 comedy is one of the enduring treasures of the movies, a film
of inspiration and meticulous craft, a movie that's about nothing but sex an
d yet pretends it's about crime and greed. It is underwired with Wilder's ch
eerful cynicism, so that no time is lost to soppiness and everyone behaves a
ccording to basic Darwinian drives. When sincere emotion strikes these chara
cters, it blindsides them: Curtis thinks he wants only sex, Monroe thinks sh
e wants only money, and they are as astonished as delighted to find they wan
t only each other.

The plot is classic screwball. Curtis and Lemmon play Chicago musicians who
disguise themselves as women to avoid being rubbed out after they witness th
e St. Valentine's Day Massacre. They join an all-girl orchestra on its way t
o Florida. Monroe is the singer, who dreams of marrying a millionaire but de
spairs, "I always get the fuzzy end of the lollipop." Curtis lusts for Monro
e and disguises himself as a millionaire to win her. Monroe lusts after mone
y and gives him lessons in love. Their relationship is flipped and mirrored
in low comedy as Lemmon gets engaged to a real millionaire, played by Joe E.
Brown. "You're not a girl!" Curtis protests to Lemmon. "You're a guy! Why wo
uld a guy want to marry a guy?" Lemmon: "Security!"

The movie has been compared to Marx Brothers classics, especially in the sla
pstick chases as gangsters pursue the heroes through hotel corridors. The we
ak points in many Marx Brothers films are the musical interludes--not Harp
o's solos, but the romantic duets involving insipid supporting characters.
"Some Like It Hot" has no problems with its musical numbers because the sing
er is Monroe, who didn't have a great singing voice but was as good as Frank
Sinatra at selling the lyrics.

Consider her solo of "I Wanna Be Loved by You." The situation is as basic as
it can be: a pretty girl standing in front of an orchestra and singing a son
g. Monroe and Wilder turn it into one of the most mesmerizing and blatantly
sexual scenes in the movies. She wears that clinging, see-through dress, gau
ze covering the upper slopes of her breasts, the neckline scooping to a cens
or's eyebrow north of trouble. Wilder places her in the center of a round sp
otlight that does not simply illuminate her from the waist up, as an ordinar
y spotlight would, but toys with her like a surrogate neckline, dipping and
clinging as Monroe moves her body higher and lower in the light with teasing
precision. It is a striptease in which nudity would have been superfluous. A
ll the time she seems unaware of the effect, singing the song innocently, as
if she thinks it's the literal truth. To experience that scene is to underst
and why no other actor, male or female, has more sexual chemistry with the c
```

```
amera than Monroe.

Capturing the chemistry was not all that simple. Legends surround "Some Like
It Hot." Kissing Marilyn, Curtis famously said, was like kissing Hitler. Mon
roe had so much trouble saying one line ("Where's the bourbon?") while looki
ng in a dresser drawer that Wilder had the line pasted inside the drawer. Th
en she opened the wrong drawer. So he had it pasted inside every drawer.

Monroe's eccentricities and neuroses on sets became notorious, but studios p
ut up with her long after any other actress would have been blackballed beca
use what they got back on the screen was magical. Watch the final take of "W
here's the bourbon?" and Monroe seems utterly spontaneous. And watch the fam
ous scene aboard the yacht, where Curtis complains that no woman can arouse
him, and Marilyn does her best. She kisses him not erotically but tenderly,
sweetly, as if offering a gift and healing a wound. You remember what Curtis
said but when you watch that scene, all you can think is that Hitler must ha
ve been a terrific kisser.

The movie is really the story of the Lemmon and Curtis characters, and it's
got a top-shelf supporting cast (Joe E. Brown, George Raft, Pat O'Brien), bu
t Monroe steals it, as she walked away with every movie she was in. It is an
act of the will to watch anyone else while she is on the screen. Tony Curti
s' performance is all the more admirable because we know how many takes she
needed--Curtis must have felt at times like he was in a pro-am tournament. Y
et he stays fresh and alive in sparkling dialogue scenes like their first me
eting on the beach, where he introduces himself as the Shell Oil heir and wi
ckedly parodies Cary Grant. Watch his timing in the yacht seduction scene, a
nd the way his character plays with her naivete. "Water polo? Isn't that ter
ribly dangerous?" asks Monroe. Curtis: "I'll say! I had two ponies drown und
er me."

Watch, too, for Wilder's knack of hiding bold sexual symbolism in plain vie
w. When Monroe first kisses Curtis while they're both horizontal on the couc
h, notice how his patent-leather shoe rises phallically in the mid-distance
behind her. Does Wilder intend this effect? Undoubtedly, because a little la
ter, after the frigid millionaire confesses he has been cured, he says, "I'v
e got a funny sensation in my toes--like someone was barbecuing them over a
slow flame." Monroe's reply: "Let's throw another log on the fire."

Jack Lemmon gets the fuzzy end of the lollipop in the parallel relationship.
The screenplay by Wilder and I.A.L. Diamond is Shakespearean in the way it c
uts between high and low comedy, between the heroes and the clowns. The Curt
is character is able to complete his round trip through gender, but Lemmon g
ets stuck halfway, so that Curtis connects with Monroe in the upstairs love
story while Lemmon is downstairs in the screwball department with Joe E. Bro
wn. Their romance is frankly cynical: Brown's character gets married and div
orced the way other men date, and Lemmon plans to marry him for the alimony.

But they both have so much fun in their courtship! While Curtis and Monroe a
re on Brown's yacht, Lemmon and Brown are dancing with such perfect timing t
hat a rose in Lemmon's teeth ends up in Brown's. Lemmon has a hilarious scen
e the morning after his big date, laying on his bed, still in drag, playing
with castanets as he announces his engagement. (Curtis: "What are you going
to do on your honeymoon?" Lemmon: "He wants to go to the Riviera, but I kind
a lean toward Niagara Falls.") Both Curtis and Lemmon are practicing cruel d
eceptions--Curtis has Monroe thinking she's met a millionaire, and Brown thi
nks Lemmon is a woman--but the film dances free before anyone gets hurt. Bot
h Monroe and Brown learn the truth and don't care, and after Lemmon reveals
he's a man, Brown delivers the best curtain line in the movies. If you've se
en the movie, you know what it is, and if you haven't, you deserve to hear i
t for the first time from him.
```

2. Tokenize your movie file and print the following

a. How many sentences in the file?

In [3]:

```
from nltk.tokenize import sent_tokenize
```

In [4]:

```
st=sent_tokenize(cont)
len(st)
```

Out[4]:

```
77
```

b. How many words in the file?

In [5]:

```
from nltk.tokenize import word_tokenize
```

In [6]:

```
tokenizer = nltk.tokenize.WhitespaceTokenizer()
tok = tokenizer.tokenize(cont)
len(tok)
```

Out[6]:

```
1289
```

c. What are the top 10 words and their counts?

In [7]:

```
tokfd=FreqDist(tok)
tokfd.most_common(10)
```

Out[7]:

```
[('the', 68),
 ('and', 39),
 ('a', 33),
 ('in', 26),
 ('of', 22),
 ('to', 22),
 ('is', 21),
 ('as', 19),
 ('Curtis', 15),
 ('Monroe', 14)]
```

d. How many different POS tags are represented in this file?

In [8]:

```
tag = []
tem = []
tok = [w for w in tok if not w in stop_words]
tagged = nltk.pos_tag(tok)
for i in tagged:
    (word,pos)=i
    tag.append(pos)

for j in tag:
    if j not in tem:
        tem.append(j)
len(tem)
```

Out[8]:

```
23
```

e. What are the top 10 POS tags and their counts?

In [9]:

```
tokpos = FreqDist(tagged)
tokpos.most_common(10)
```

Out[9]:

```
[(('Curtis', 'NNP'), 15),
 (('Monroe', 'NNP'), 14),
 (('Lemmon', 'NNP'), 13),
 (('The', 'DT'), 7),
 (('It', 'PRP'), 6),
 (('like', 'IN'), 6),
 (('She', 'PRP'), 5),
 (('gets', 'VBZ'), 5),
 (('Wilder', 'NNP'), 5),
 (('seems', 'VBZ'), 4)]
```

f. How many nouns in the file?

In [10]:

```
n=0
for i in tokpos.keys():
    (word,pos)=i
    if pos == 'NN' or pos == 'NNS' or pos == 'NNP' or pos == 'NNPS':
        n+=1
print(n)
```

```
279
```

g. How many verbs in the file?

In [11]:

```
v=0
for i in tokpos.keys():
    (word,pos)=i
    if pos == 'VB' or pos == 'VBD' or pos == 'VBN' or pos == 'VBP' or pos == 'VBZ':
        v+=1
print(v)
```

108

h. How many adjectives in the file?

In [12]:

```
adj = []
for i in tokpos.keys():
    (word,pos)=i
    if pos == 'JJ' or pos == 'JJR' or pos == 'JJS':
        adj.append(i)
len(adj)
```

Out[12]:

115

i. How many adverbs in the file?

In [13]:

```
adv=[]
for i in tokpos.keys():
    (word,pos)=i
    if pos == 'RB' or pos == 'RBR' or pos == 'RBS' or pos == 'BP':
        adv.append(i)
len(adv)
```

Out[13]:

29

j. What is the most frequent adverb?

In [14]:

```
adv = FreqDist(adv)
adv.most_common(1)
```

Out[14]:

[(('still', 'RB'), 1)]

k. What is the most frequent adjective?

In [15]:

```
adj = FreqDist(adj)
adj.most_common(1)
```

Out[15]:

```
[(('icon,', 'JJ'), 1)]
```

205229133 ¶

Natural Language Processing Lab

Lab9. Building Bigram Tagger

In [1]:

```
import nltk
from nltk.tokenize import sent_tokenize, word_tokenize
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\ELCOT\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\ELCOT\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

Out[1]:

```
True
```

EXERCISE-1

In [2]:

```
import nltk
text = word_tokenize("And now for something completely different")
nltk.pos_tag(text)
```

Out[2]:

```
[('And', 'CC'),
 ('now', 'RB'),
 ('for', 'IN'),
 ('something', 'NN'),
 ('completely', 'RB'),
 ('different', 'JJ')]
```

EXERCISE-2

In [3]:

```
from nltk.corpus import brown
tagsen = brown.tagged_sents()
```

STEP 1: Prepare data sets

In [4]:

```
len(tagsen)
```

Out[4]:

```
57340
```

In [5]:

```
br_train=tagsen[0:50000]
br_test=tagsen[50000:]
br_test[0]
```

Out[5]:

```
[('I', 'PPSS'),
 ('was', 'BEDZ'),
 ('loaded', 'VBN'),
 ('with', 'IN'),
 ('suds', 'NNS'),
 ('when', 'WRB'),
 ('I', 'PPSS'),
 ('ran', 'VBD'),
 ('away', 'RB'),
 (',', ','),
 ('and', 'CC'),
 ('I', 'PPSS'),
 ("haven't", 'HV*'),
 ('had', 'HVN'),
 ('a', 'AT'),
 ('chance', 'NN'),
 ('to', 'TO'),
 ('wash', 'VB'),
 ('it', 'PPO'),
 ('off', 'RP'),
 ('.', '.')]
```

STEP 2: Build a bigram tagger

In [6]:

```
t0 = nltk.DefaultTagger('NN')
t1 = nltk.UnigramTagger(br_train, backoff=t0)
t2 = nltk.BigramTagger(br_train, backoff=t1)
```

STEP 3: Evaluate

In [7]:

```
t2.evaluate(br_test)
```

Out[7]:

```
0.9111006662708622
```

STEP 4: Explore

1. How big are your training data and testing data? Answer in terms of the number of total words in them.

In [8]:

```
total_train=[len(l) for l in br_train]
sum(total_train)
```

Out[8]:

```
1039920
```

In [9]:

```
total_test=[len(l) for l in br_test]
sum(total_test)
```

Out[9]:

```
121272
```

2. What is the performance of each of the two back-off taggers? How much improvement did you get: (1) going from the default tagger to the unigram tagger, and (2) going from the unigram tagger to the bigram tagger?

In [10]:

```
t1.evaluate(br_test)
```

Out[10]:

```
0.8897849462365591
```

In [11]:

```
t2.evaluate(br_test)
```

Out[11]:

```
0.9111006662708622
```

3. Recall that 'cold' is ambiguous between JJ 'adjective' and NN 'singular noun'. Let's explore the word in the training data. The problem with the training data, through, is that it is a list of tagged sentences, and it's difficult to get to the tagged words which are one level below

In [12]:

```
br_train[0]
```

Out[12]:

```
[('The', 'AT'),
 ('Fulton', 'NP-TL'),
 ('County', 'NN-TL'),
 ('Grand', 'JJ-TL'),
 ('Jury', 'NN-TL'),
 ('said', 'VBD'),
 ('Friday', 'NR'),
 ('an', 'AT'),
 ('investigation', 'NN'),
 ('of', 'IN'),
 ("Atlanta's", 'NP$'),
 ('recent', 'JJ'),
 ('primary', 'NN'),
 ('election', 'NN'),
 ('produced', 'VBD'),
 ('``', '``'),
 ('no', 'AT'),
 ('evidence', 'NN'),
 ("''", "''"),
 ('that', 'CS'),
 ('any', 'DTI'),
 ('irregularities', 'NNS'),
 ('took', 'VBD'),
 ('place', 'NN'),
 ('.', '.')]
```

In [13]:

```
br_train[1277]
```

Out[13]:

```
[('``', '``'),
 ('I', 'PPSS'),
 ('told', 'VBD'),
 ('him', 'PPO'),
 ('who', 'WPS'),
 ('I', 'PPSS'),
 ('was', 'BEDZ'),
 ('and', 'CC'),
 ('he', 'PPS'),
 ('was', 'BEDZ'),
 ('quite', 'QL'),
 ('cold', 'JJ'),
 ('.', '.')]
```

In [14]:

```
br_train[1277][11]
```

Out[14]:

```
('cold', 'JJ')
```

4. To be able to compile tagged-word-level statistics, we will need a flat list of

tagged words,without them being organized into sentences. How to do this? You can use multi-loop list comprehension to construct it:

In [15]:

```
br_train_flat = [(word, tag) for sent in br_train for (word, tag) in sent]
```

In [16]:

```
br_train_flat[:40]
```

Out[16]:

```
[('The', 'AT'),
 ('Fulton', 'NP-TL'),
 ('County', 'NN-TL'),
 ('Grand', 'JJ-TL'),
 ('Jury', 'NN-TL'),
 ('said', 'VBD'),
 ('Friday', 'NR'),
 ('an', 'AT'),
 ('investigation', 'NN'),
 ('of', 'IN'),
 ("Atlanta's", 'NP$'),
 ('recent', 'JJ'),
 ('primary', 'NN'),
 ('election', 'NN'),
 ('produced', 'VBD'),
 ('``', '``'),
 ('no', 'AT'),
 ('evidence', 'NN'),
 ("''", "''"),
 ('that', 'CS'),
 ('any', 'DTI'),
 ('irregularities', 'NNS'),
 ('took', 'VBD'),
 ('place', 'NN'),
 ('.', '.'),
 ('The', 'AT'),
 ('jury', 'NN'),
 ('further', 'RBR'),
 ('said', 'VBD'),
 ('in', 'IN'),
 ('term-end', 'NN'),
 ('presentments', 'NNS'),
 ('that', 'CS'),
 ('the', 'AT'),
 ('City', 'NN-TL'),
 ('Executive', 'JJ-TL'),
 ('Committee', 'NN-TL'),
 (',', ','),
 ('which', 'WDT'),
 ('had', 'HVD')]
```

In [17]:

```
br_train_flat[13]
```

Out[17]:

```
('election', 'NN')
```

5. Now, exploring this list of (word, POS) pairs from the training data, answer the questions below.

a. Which is the more likely tag for 'cold' overall?

In [18]:

```
fd = nltk.FreqDist(br_train_flat)
cfd = nltk.ConditionalFreqDist(br_train_flat)
```

In [19]:

```
cfd['cold'].most_common()
```

Out[19]:

```
[('JJ', 110), ('NN', 8), ('RB', 2)]
```

b. When the POS tag of the preceding word (call it POSn-1) is AT, what is the likelihood of 'cold' being a noun? How about it being an adjective?

In [20]:

```
br_train_2grams = list(nltk.ngrams(br_train_flat,2))
br_train_cold=[a[1] for (a, b) in br_train_2grams if b[0] == 'cold']
fdist = nltk.FreqDist(br_train_cold)
[tag for (tag, _) in fdist.most_common()]
```

Out[20]:

```
['AT',
 'IN',
 'CC',
 'QL',
 'BEDZ',
 'JJ',
 ',',
 'DT',
 'PP$',
 'RP',
 '``',
 'NN',
 'VBN',
 'VBD',
 'CS',
 'BEZ',
 'DOZ',
 'RB',
 'PPSS',
 'BE',
 'VB',
 'VBZ',
 'NP$',
 'BEDZ*',
 '--',
 'DTI',
 'WRB',
 'BED']
```

c. When POSn-1 is JJ, what is the likelihood of 'cold' being a noun? How about it being an adjective?

In [21]:

```
br_pre = [(w2+"/"+t2, t1) for ((w1,t1),(w2,t2)) in br_train_2grams]
br_pre_cfd = nltk.ConditionalFreqDist(br_pre)
br_pre
```

Out[21]:

```
[('Fulton/NP-TL', 'AT'),
 ('County/NN-TL', 'NP-TL'),
 ('Grand/JJ-TL', 'NN-TL'),
 ('Jury/NN-TL', 'JJ-TL'),
 ('said/VBD', 'NN-TL'),
 ('Friday/NR', 'VBD'),
 ('an/AT', 'NR'),
 ('investigation/NN', 'AT'),
 ('of/IN', 'NN'),
 ("Atlanta's/NP$", 'IN'),
 ('recent/JJ', 'NP$'),
 ('primary/NN', 'JJ'),
 ('election/NN', 'NN'),
 ('produced/VBD', 'NN'),
 ('``/``', 'VBD'),
 ('no/AT', '``'),
 ('evidence/NN', 'AT'),
 ("''/''", 'NN'),
```

d. Can you find any POSn-1 that favors NN over JJ for the following word 'cold'?

In [22]:

```
br_pre_cfd['cold/NN'].most_common()
```

Out[22]:

```
[('AT', 4), ('JJ', 2), (',', 1), ('DT', 1)]
```

In [23]:

```
br_pre_cfd['cold/JJ'].most_common()
```

Out[23]:

```
[('AT', 38),
 ('IN', 14),
 ('CC', 8),
 ('QL', 7),
 ('BEDZ', 7),
 ('JJ', 4),
 ('DT', 3),
 (',', 3),
 ('PP$', 3),
 ('``', 2),
 ('NN', 2),
 ('VBN', 2),
 ('VBD', 2),
 ('CS', 1),
 ('BEZ', 1),
 ('DOZ', 1),
 ('RB', 1),
 ('PPSS', 1),
 ('BE', 1),
 ('VB', 1),
 ('VBZ', 1),
 ('NP$', 1),
 ('BEDZ*', 1),
 ('--', 1),
 ('RP', 1),
 ('DTI', 1),
 ('WRB', 1),
 ('BED', 1)]
```

6. Based on what you found, how is your bigram tagger expected to tag 'cold' in the following sentences?

In [24]:

```
bigram_tagger = nltk.BigramTagger(br_train)
```

a. I was very cold.

In [25]:

```
text1 = word_tokenize("I was very cold.")
bigram_tagger.tag(text1)
```

Out[25]:

```
[('I', 'PPSS'), ('was', 'BEDZ'), ('very', 'QL'), ('cold', 'JJ'), ('.', '.')]
```

b. I had a cold.

In [26]:

```
text2 = word_tokenize("I had cold.")
bigram_tagger.tag(text2)
```

Out[26]:

```
[('I', 'PPSS'), ('had', 'HVD'), ('cold', None), ('.', None)]
```

c. I had a severe cold.

In [27]:

```
text3 = word_tokenize("I had a severe cold.")
bigram_tagger.tag(text3)
```

Out[27]:

```
[('I', 'PPSS'),
 ('had', 'HVD'),
 ('a', 'AT'),
 ('severe', 'JJ'),
 ('cold', 'JJ'),
 ('.', '.')]
```

d. January was a cold month.

In [28]:

```
text4 = word_tokenize("January was a cold month")
bigram_tagger.tag(text4)
```

Out[28]:

```
[('January', None),
 ('was', None),
 ('a', None),
 ('cold', None),
 ('month', None)]
```

7. Verify your prediction by having the tagger actually tag the four sentences. What did you find?

In [ ]:

```

```

8. Have the tagger tag the following sentences, all of which contain the word 'so':

a. I failed to do so.

In [29]:

```
text5 = word_tokenize("I failed to do so")
bigram_tagger.tag(text5)
```

Out[29]:

```
[('I', 'PPSS'), ('failed', 'VBD'), ('to', 'TO'), ('do', 'DO'), ('so', 'RB')]
```

b. I was happy, but so was my enemy

In [30]:

```
text6 = word_tokenize("I was happy,but so was my enemy")
bigram_tagger.tag(text6)
```

Out[30]:

```
[('I', 'PPSS'),
 ('was', 'BEDZ'),
 ('happy', 'JJ'),
 (',', ','),
 ('but', 'CC'),
 ('so', 'RB'),
 ('was', 'BEDZ'),
 ('my', 'PP$'),
 ('enemy', 'NN')]
```

c. So, how was the exam?

In [31]:

```
text7 = word_tokenize("So, how was the exam?")
bigram_tagger.tag(text7)
```

Out[31]:

```
[('So', 'RB'),
 (',', ','),
 ('how', 'WRB'),
 ('was', 'BEDZ'),
 ('the', 'AT'),
 ('exam', None),
 ('?', None)]
```

d. The students came in early so they can get good seats.

In [32]:

```
text8 = word_tokenize("The students came in early so they can get good seats")
bigram_tagger.tag(text8)
```

Out[32]:

```
[('The', 'AT'),
 ('students', 'NNS'),
 ('came', 'VBD'),
 ('in', 'IN'),
 ('early', 'JJ'),
 ('so', 'CS'),
 ('they', 'PPSS'),
 ('can', 'MD'),
 ('get', 'VB'),
 ('good', 'JJ'),
 ('seats', 'NNS')]
```

e. She failed the exam, so she must take it again.

In [33]:

```
text9 = word_tokenize("She failed the exam, so she must take it again")
bigram_tagger.tag(text9)
```

Out[33]:

```
[('She', 'PPS'),
 ('failed', 'VBD'),
 ('the', 'AT'),
 ('exam', None),
 (',', None),
 ('so', None),
 ('she', None),
 ('must', None),
 ('take', None),
 ('it', None),
 ('again', None)]
```

f. That was so incredible.

In [34]:

```
text10 = word_tokenize("That was so incredible")
bigram_tagger.tag(text10)
```

Out[34]:

```
[('That', 'DT'), ('was', 'BEDZ'), ('so', 'QL'), ('incredible', 'JJ')]
```

g. Wow, so incredible.

In [35]:

```
text11 = word_tokenize("Wow, so incredible")
bigram_tagger.tag(text11)
```

Out[35]:

```
[('Wow', None), (',', None), ('so', None), ('incredible', None)]
```

9. Examine the tagger's performance on the sentences, focusing on the word 'so'. For each of them, decide if the tagger's output is correct, and explain how the tagger determined the POS tag.

In [ ]:

```

```

10.Based on what you have observed so far, offer a critique on the bigram tagger. What are its strengths and what are its limitations?

In [ ]:

```

```

Lab10_NLP_viviyan

May 26, 2021

0.0.1 viviyan richards w

205229133

In this lab, you will extract named entities from the given text file using NLTK. You will also recognize entities based on the regular expression patterns.

0.0.2 EXERCISE-1

0.0.3 Extract all named entities from the following text:

[1]:
```
import nltk
from nltk.tree import Tree
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag
from nltk.chunk import ne_chunk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\RAVIKUMAR\AppData\Roaming\nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\RAVIKUMAR\AppData\Roaming\nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     C:\Users\RAVIKUMAR\AppData\Roaming\nltk_data…
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words to
[nltk_data]     C:\Users\RAVIKUMAR\AppData\Roaming\nltk_data…
[nltk_data]   Package words is already up-to-date!
```

[1]: True

[2]:

```
Sentence1 = "Rajkumar said on Monday that WASHINGTON -- In the wake of a string␣
↪of abuses by New York police officers in the 1990s,Loretta E. Lynch, the top␣
↪federal prosecutor in Brooklyn, spoke forcefully about the pain of a broken␣
↪trust that African-Americans felt and said the responsibility for repairing␣
↪generations of miscommunication and mistrust fell to law enforcement."
```

[3]:
```
tokens = word_tokenize(Sentence1)
tags = pos_tag(tokens)
ne_tree = ne_chunk(tags)
print(ne_tree[:])
```

```
[Tree('PERSON', [('Rajkumar', 'NNP')]), ('said', 'VBD'), ('on', 'IN'),
('Monday', 'NNP'), ('that', 'IN'), Tree('ORGANIZATION', [('WASHINGTON',
'NNP')]), ('--', ':'), ('In', 'IN'), ('the', 'DT'), ('wake', 'NN'), ('of',
'IN'), ('a', 'DT'), ('string', 'NN'), ('of', 'IN'), ('abuses', 'NNS'), ('by',
'IN'), Tree('GPE', [('New', 'NNP'), ('York', 'NNP')]), ('police', 'NN'),
('officers', 'NNS'), ('in', 'IN'), ('the', 'DT'), ('1990s', 'CD'), (',', ','),
Tree('PERSON', [('Loretta', 'NNP'), ('E.', 'NNP'), ('Lynch', 'NNP')]), (',',
','), ('the', 'DT'), ('top', 'JJ'), ('federal', 'JJ'), ('prosecutor', 'NN'),
('in', 'IN'), Tree('GPE', [('Brooklyn', 'NNP')]), (',', ','), ('spoke', 'VBD'),
('forcefully', 'RB'), ('about', 'IN'), ('the', 'DT'), ('pain', 'NN'), ('of',
'IN'), ('a', 'DT'), ('broken', 'JJ'), ('trust', 'NN'), ('that', 'IN'),
('African-Americans', 'NNP'), ('felt', 'VBD'), ('and', 'CC'), ('said', 'VBD'),
('the', 'DT'), ('responsibility', 'NN'), ('for', 'IN'), ('repairing', 'VBG'),
('generations', 'NNS'), ('of', 'IN'), ('miscommunication', 'NN'), ('and', 'CC'),
('mistrust', 'NN'), ('fell', 'VBD'), ('to', 'TO'), ('law', 'NN'),
('enforcement', 'NN'), ('.', '.')]
```

[4]:
```
ne_tree = ne_chunk(pos_tag(word_tokenize(Sentence1)))
```

[5]:
```
for i in ne_tree:
    print(i)
```

```
(PERSON Rajkumar/NNP)
('said', 'VBD')
('on', 'IN')
('Monday', 'NNP')
('that', 'IN')
(ORGANIZATION WASHINGTON/NNP)
('--', ':')
('In', 'IN')
('the', 'DT')
('wake', 'NN')
('of', 'IN')
('a', 'DT')
('string', 'NN')
('of', 'IN')
('abuses', 'NNS')
```

```
('by', 'IN')
(GPE New/NNP York/NNP)
('police', 'NN')
('officers', 'NNS')
('in', 'IN')
('the', 'DT')
('1990s', 'CD')
(',', ',')
(PERSON Loretta/NNP E./NNP Lynch/NNP)
(',', ',')
('the', 'DT')
('top', 'JJ')
('federal', 'JJ')
('prosecutor', 'NN')
('in', 'IN')
(GPE Brooklyn/NNP)
(',', ',')
('spoke', 'VBD')
('forcefully', 'RB')
('about', 'IN')
('the', 'DT')
('pain', 'NN')
('of', 'IN')
('a', 'DT')
('broken', 'JJ')
('trust', 'NN')
('that', 'IN')
('African-Americans', 'NNP')
('felt', 'VBD')
('and', 'CC')
('said', 'VBD')
('the', 'DT')
('responsibility', 'NN')
('for', 'IN')
('repairing', 'VBG')
('generations', 'NNS')
('of', 'IN')
('miscommunication', 'NN')
('and', 'CC')
('mistrust', 'NN')
('fell', 'VBD')
('to', 'TO')
('law', 'NN')
('enforcement', 'NN')
('.', '.')
```

0.0.4 Question-1

0.0.5 Count and print the number of PERSON, LOCATION and ORGANIZATION in the given sentence.

[6]:
```
import nltk
from collections import Counter
for chunk in ne_tree:
    if hasattr(chunk, 'label'):
        print([Counter(label) for label in chunk])
```

```
[Counter({'Rajkumar': 1, 'NNP': 1})]
[Counter({'WASHINGTON': 1, 'NNP': 1})]
[Counter({'New': 1, 'NNP': 1}), Counter({'York': 1, 'NNP': 1})]
[Counter({'Loretta': 1, 'NNP': 1}), Counter({'E.': 1, 'NNP': 1}),
Counter({'Lynch': 1, 'NNP': 1})]
[Counter({'Brooklyn': 1, 'NNP': 1})]
```

0.1 Question 2

0.1.1 Observe the results. Does named entity, "police officers" get recognized?.

[7]:
```
word = nltk.word_tokenize(Sentence1)
pos_tag = nltk.pos_tag(word)
chunk = nltk.ne_chunk(pos_tag)
grammar = "NP: {<NN><NNS>}"
cp = nltk.RegexpParser(grammar)
result = cp.parse(chunk)
NE = [ " ".join(w for w, t in ele) for ele in result if isinstance(ele, nltk.
 ↪Tree)]
print (NE)
```

```
['Rajkumar', 'WASHINGTON', 'New York', 'police officers', 'Loretta E. Lynch',
'Brooklyn']
```

0.1.2 Write a regular expression patter to detect this. You will need nltk.RegexpParser class to define pattern and parse terms to detect patterns.

[8]:
```
grammar = "NP: {<NN><NNS>}"
cp = nltk.RegexpParser(grammar)
result = cp.parse(ne_tree)
NE = [ " ".join(w for w, t in ele) for ele in result if isinstance(ele, nltk.
 ↪Tree)]
print(NE)
```

```
['Rajkumar', 'WASHINGTON', 'New York', 'police officers', 'Loretta E. Lynch',
'Brooklyn']
```

0.1.3 Question-3

Does the named entity, "the top federal prosecutor" get recognized?.

[9]:
```
out=cp.parse(tags)
print(out[:])
```

```
[('Rajkumar', 'NNP'), ('said', 'VBD'), ('on', 'IN'), ('Monday', 'NNP'), ('that',
'IN'), ('WASHINGTON', 'NNP'), ('--', ':'), ('In', 'IN'), ('the', 'DT'), ('wake',
'NN'), ('of', 'IN'), ('a', 'DT'), ('string', 'NN'), ('of', 'IN'), ('abuses',
'NNS'), ('by', 'IN'), ('New', 'NNP'), ('York', 'NNP'), Tree('NP', [('police',
'NN'), ('officers', 'NNS')]), ('in', 'IN'), ('the', 'DT'), ('1990s', 'CD'),
(',', ','), ('Loretta', 'NNP'), ('E.', 'NNP'), ('Lynch', 'NNP'), (',', ','),
('the', 'DT'), ('top', 'JJ'), ('federal', 'JJ'), ('prosecutor', 'NN'), ('in',
'IN'), ('Brooklyn', 'NNP'), (',', ','), ('spoke', 'VBD'), ('forcefully', 'RB'),
('about', 'IN'), ('the', 'DT'), ('pain', 'NN'), ('of', 'IN'), ('a', 'DT'),
('broken', 'JJ'), ('trust', 'NN'), ('that', 'IN'), ('African-Americans', 'NNP'),
('felt', 'VBD'), ('and', 'CC'), ('said', 'VBD'), ('the', 'DT'),
('responsibility', 'NN'), ('for', 'IN'), ('repairing', 'VBG'), ('generations',
'NNS'), ('of', 'IN'), ('miscommunication', 'NN'), ('and', 'CC'), ('mistrust',
'NN'), ('fell', 'VBD'), ('to', 'TO'), ('law', 'NN'), ('enforcement', 'NN'),
('.', '.')]
```

Write a regular expression pattern to detect this.

[10]:
```
grammar = "NP: {<DT><JJ>*<NN>}"
cp = nltk.RegexpParser(grammar)
result = cp.parse(ne_tree)
NE = [ " ".join(w for w, t in ele) for ele in result if isinstance(ele, nltk.
 ↪Tree)]
print (NE)
```

```
['Rajkumar', 'WASHINGTON', 'the wake', 'a string', 'New York', 'Loretta E.
Lynch', 'the top federal prosecutor', 'Brooklyn', 'the pain', 'a broken trust',
'the responsibility']
```

0.2 EXERCISE-2

0.2.1 Question-1

0.2.2 Observe the output. Does your code recognize the NE shown in BOLD?($5.1 billion, the mobile phone, the company)

[16]:
```
Sentence2 = "European authorities fined Google a record $5.1 billion on␣
 ↪Wednesday for abusing its power in the mobile phone market and ordered the␣
 ↪company to alter its practices"
```

[24]:
```
tok = word_tokenize(Sentence2)
tagged = nltk.pos_tag(tok)
ne_tree2 = nltk.ne_chunk(tagged,binary=False)
```

```
print(ne_tree2[:])
```

```
[Tree('GPE', [('European', 'JJ')]), ('authorities', 'NNS'), ('fined', 'VBD'),
Tree('PERSON', [('Google', 'NNP')]), ('a', 'DT'), ('record', 'NN'), ('$', '$'),
('5.1', 'CD'), ('billion', 'CD'), ('on', 'IN'), ('Wednesday', 'NNP'), ('for',
'IN'), ('abusing', 'VBG'), ('its', 'PRP$'), ('power', 'NN'), ('in', 'IN'),
('the', 'DT'), ('mobile', 'JJ'), ('phone', 'NN'), ('market', 'NN'), ('and',
'CC'), ('ordered', 'VBD'), ('the', 'DT'), ('company', 'NN'), ('to', 'TO'),
('alter', 'VB'), ('its', 'PRP$'), ('practices', 'NNS')]
```

0.2.3 Write a regular expression that recognizes the entity

[52]:
```
word = nltk.word_tokenize(Sentence2)
pos_tag = nltk.pos_tag(word)
chunk = nltk.ne_chunk(pos_tag)
grammar = "NP: {<CD>|<DT><JJ>*<NN>}"
cp = nltk.RegexpParser(grammar)
result = cp.parse(chunk)
NE = [ " ".join(w for w, t in ele) for ele in result if isinstance(ele, nltk.
 ↪Tree)]
print (NE)
```

```
['European', 'Google', 'a record', '5.1', 'billion', 'the mobile phone', 'the
company']
```

0.2.4 Question-2

0.2.5 Write a regular expression that recognizes the entity, "the mobile phone" and similar to this entity such as "the company

[50]:
```
word = nltk.word_tokenize(Sentence2)
pos_tag = nltk.pos_tag(word)
chunk = nltk.ne_chunk(pos_tag)
grammar = "NP: {<DT><JJ>*<NN>}"
cp = nltk.RegexpParser(grammar)
result = cp.parse(chunk)
NE = [ " ".join(w for w, t in ele) for ele in result if isinstance(ele, nltk.
 ↪Tree)]
print (NE)
```

```
['European', 'Google', 'a record', 'the mobile phone', 'the company']
```

Name :Viviyan Richards W

Roll no:205229133

In [1]:
```
import nltk,re,pprint
from nltk.tree import Tree
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag
from nltk.chunk import ne_chunk
import numpy as npt
```

Exercise 1

In [2]:
```
np =nltk.Tree.fromstring('(NP (N Marge))')
np.pretty_print()
```

```
  NP
  |
  N
  |
Marge
```

In [3]:
```
vp = nltk.Tree.fromstring('(VP (V make) (NP (DET a) (N ham) (N sandwich)))')
vp.pretty_print()
```

```
         VP
  _______|___
 |           NP
 |     ______|_____
 V   DET     N     N
 |    |      |     |
make  a     ham sandwich
```

In [4]:
```
aux= nltk.Tree.fromstring('(AUX will)')
aux.pretty_print()
```

```
AUX
 |
will
```

Excercise 2

In [ ]:

Exercise 3

In [5]:
```
s1 = nltk.Tree.fromstring('(S (NP (N Marge)) (AUX will) (VP (V make) (NP (DET a)
s1.pretty_print()
```

```
            S
   _________|________
  |    |             VP
  |    |     ________|___
  NP   |    |            NP
  |    |    |     _______|_____
  N   AUX   V   DET      N     N
  |    |    |    |       |     |
Marge will make  a      ham sandwich
```

In [6]:
```
s2 = nltk.Tree.fromstring('(S (AUX will) (NP (N Marge)) (VP (V make) (NP (DET a)
s2.pretty_print()
```

```
            S
  __________|________
 |     |             VP
 |     |     ________|___
 |     NP   |            NP
 |     |    |     _______|_____
AUX    N    V   DET      N     N
 |     |    |    |       |     |
will Marge make  a      ham sandwich
```

Exercise 4

In [7]:
```
s3 = nltk.Tree.fromstring('(S (NP (N Homer)) (VP (V ate) (NP (DET the) (N donut))
s3.pretty_print()
```

```
           S
   ________|___
  |            VP
  |     _______|___
  NP   |           NP
  |    |        ___|____
  N    V      DET       N
  |    |       |        |
Homer ate      the      donut
```

Exercise 5

In [8]:
```python
s4 = nltk.Tree.fromstring('(S (NP (DET my)(ADJ old)(N cat))(VP(VP(V died))(PP(P c
s4.pretty_print()
```

```
         S
      ___|________
     |            VP
     |        ____|___
     |       |        PP
     |       |     ___|_____
     NP      VP   |         NP
  ___|___    |    |         |
DET ADJ  N   V    P         N
 |   |   |   |    |         |
 my old cat died  on      Tuesday
```

In [9]:
```python
s5 = nltk.Tree.fromstring('(S(NP (N children))(AUX must)(VP(VP(V play))(PP(P in)(
s5.pretty_print()
```

```
                        S
    ____________________|_______
   |      |                     VP
   |      |     ________________|__________
   |      |    |        PP                 PP
   |      |    |     ___|___          _____|____
   NP     |    VP   |       NP       |          NP
   |      |    |    |    ___|___     |      ____|_____
   N     AUX   V    P  DET      N    P   DET         N
   |      |    |    |   |       |    |     |          |
children must play  in the     park with their     friends
```

Exercise 6

In [10]:
```python
print(vp) #this is from exercise 1
```

```
(VP (V make) (NP (DET a) (N ham) (N sandwich)))
```

In [11]:
```python
vp_rules = vp.productions()
vp_rules
```

Out[11]:
```
[VP -> V NP,
 V -> 'make',
 NP -> DET N N,
 DET -> 'a',
 N -> 'ham',
 N -> 'sandwich']
```

In [12]:
```
vp_rules[0]
```

Out[12]:
```
VP -> V NP
```

In [13]:
```
vp_rules[1]
```

Out[13]:
```
V -> 'make'
```

In [14]:
```
vp_rules[0].is_lexical()
```

Out[14]:
```
False
```

In [15]:
```
vp_rules[0].is_lexical()
```

Out[15]:
```
False
```

Explore the CF rules of s5

In [16]:
```
print(s5)
```

```
(S
  (NP (N children))
  (AUX must)
  (VP
    (VP (V play))
    (PP (P in) (NP (DET the) (N park)))
    (PP (P with) (NP (DET their) (N friends)))))
```

In [17]:
```
s5_rules=s5.productions()
s5_rules
```

Out[17]:
```
[S -> NP AUX VP,
 NP -> N,
 N -> 'children',
 AUX -> 'must',
 VP -> VP PP PP,
 VP -> V,
 V -> 'play',
 PP -> P NP,
 P -> 'in',
 NP -> DET N,
 DET -> 'the',
 N -> 'park',
 PP -> P NP,
 P -> 'with',
 NP -> DET N,
 DET -> 'their',
 N -> 'friends']
```

a. How many CF rules are used in s5?

In [18]:
```
print("How Many CF values are used in s5   ",len(s5_rules))
```

```
How Many CF values are used in s5    17
```

b. How many unique CF rules are used in s5?

In [19]:
```
x = npt.array(s5_rules)
print("How Many unique CF rules are used in s5  ",len(npt.unique(x)))
```

```
How Many unique CF rules are used in s5   15
```

c. How many of them are lexical?

In [20]:
```
n=0
for x in s5_rules:
  if x.is_lexical():
    n = n+1
print("How many of them are lexical?  ",n)
```

```
How many of them are lexical?   9
```

lab12_nlp_viviyan_33

June 8, 2021

0.1 Lab12. Building and Parsing Context Free Grammars

[1]:

```
import nltk
nltk.download("punkt")
from nltk.tree import Tree
from nltk.tokenize import word_tokenize
from IPython.display import display
import nltk,re,pprint
from nltk.tag import pos_tag
from nltk.chunk import ne_chunk
import numpy as npt

!apt-get install -y xvfb # Install X Virtual Frame Buffer
import os
os.system('Xvfb :1 -screen 0 1600x1200x16  &')# create virtual display with␣
 ↪size 1600x1200 and 16 bit color. Color can be changed to 24 or 8
os.environ['DISPLAY']=':1.0'# tell X clients to use our virtual DISPLAY :1.0.
%matplotlib inline
### INSTALL GHOSTSCRIPT (Required to display NLTK trees)
!apt install ghostscript python3-tk
```

```
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt.zip.
Reading package lists… Done
Building dependency tree
Reading state information… Done
The following package was automatically installed and is no longer required:
  libnvidia-common-460
Use 'apt autoremove' to remove it.
The following NEW packages will be installed:
  xvfb
0 upgraded, 1 newly installed, 0 to remove and 34 not upgraded.
Need to get 784 kB of archives.
After this operation, 2,270 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu bionic-updates/universe amd64 xvfb amd64
2:1.19.6-1ubuntu4.9 [784 kB]
Fetched 784 kB in 1s (958 kB/s)
Selecting previously unselected package xvfb.
```

```
(Reading database … 160706 files and directories currently installed.)
Preparing to unpack …/xvfb_2%3a1.19.6-1ubuntu4.9_amd64.deb …
Unpacking xvfb (2:1.19.6-1ubuntu4.9) …
Setting up xvfb (2:1.19.6-1ubuntu4.9) …
Processing triggers for man-db (2.8.3-2ubuntu0.1) …
Reading package lists… Done
Building dependency tree
Reading state information… Done
python3-tk is already the newest version (3.6.9-1~18.04).
The following package was automatically installed and is no longer required:
  libnvidia-common-460
Use 'apt autoremove' to remove it.
The following additional packages will be installed:
  fonts-droid-fallback fonts-noto-mono gsfonts libcupsfilters1 libcupsimage2
  libgs9 libgs9-common libijs-0.35 libjbig2dec0 poppler-data
Suggested packages:
  fonts-noto ghostscript-x poppler-utils fonts-japanese-mincho
  | fonts-ipafont-mincho fonts-japanese-gothic | fonts-ipafont-gothic
  fonts-arphic-ukai fonts-arphic-uming fonts-nanum
The following NEW packages will be installed:
  fonts-droid-fallback fonts-noto-mono ghostscript gsfonts libcupsfilters1
  libcupsimage2 libgs9 libgs9-common libijs-0.35 libjbig2dec0 poppler-data
0 upgraded, 11 newly installed, 0 to remove and 34 not upgraded.
Need to get 14.1 MB of archives.
After this operation, 49.9 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu bionic/main amd64 fonts-droid-fallback
all 1:6.0.1r16-1.1 [1,805 kB]
Get:2 http://archive.ubuntu.com/ubuntu bionic/main amd64 poppler-data all
0.4.8-2 [1,479 kB]
Get:3 http://archive.ubuntu.com/ubuntu bionic/main amd64 fonts-noto-mono all
20171026-2 [75.5 kB]
Get:4 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libcupsimage2
amd64 2.2.7-1ubuntu2.8 [18.6 kB]
Get:5 http://archive.ubuntu.com/ubuntu bionic/main amd64 libijs-0.35 amd64
0.35-13 [15.5 kB]
Get:6 http://archive.ubuntu.com/ubuntu bionic/main amd64 libjbig2dec0 amd64
0.13-6 [55.9 kB]
Get:7 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libgs9-common
all 9.26~dfsg+0-0ubuntu0.18.04.14 [5,092 kB]
Get:8 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libgs9 amd64
9.26~dfsg+0-0ubuntu0.18.04.14 [2,265 kB]
Get:9 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 ghostscript
amd64 9.26~dfsg+0-0ubuntu0.18.04.14 [51.3 kB]
Get:10 http://archive.ubuntu.com/ubuntu bionic/main amd64 gsfonts all
1:8.11+urwcyr1.0.7~pre44-4.4 [3,120 kB]
Get:11 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64
libcupsfilters1 amd64 1.20.2-0ubuntu3.1 [108 kB]
Fetched 14.1 MB in 1s (10.5 MB/s)
```

```
Selecting previously unselected package fonts-droid-fallback.
(Reading database … 160713 files and directories currently installed.)
Preparing to unpack …/00-fonts-droid-fallback_1%3a6.0.1r16-1.1_all.deb …
Unpacking fonts-droid-fallback (1:6.0.1r16-1.1) …
Selecting previously unselected package poppler-data.
Preparing to unpack …/01-poppler-data_0.4.8-2_all.deb …
Unpacking poppler-data (0.4.8-2) …
Selecting previously unselected package fonts-noto-mono.
Preparing to unpack …/02-fonts-noto-mono_20171026-2_all.deb …
Unpacking fonts-noto-mono (20171026-2) …
Selecting previously unselected package libcupsimage2:amd64.
Preparing to unpack …/03-libcupsimage2_2.2.7-1ubuntu2.8_amd64.deb …
Unpacking libcupsimage2:amd64 (2.2.7-1ubuntu2.8) …
Selecting previously unselected package libijs-0.35:amd64.
Preparing to unpack …/04-libijs-0.35_0.35-13_amd64.deb …
Unpacking libijs-0.35:amd64 (0.35-13) …
Selecting previously unselected package libjbig2dec0:amd64.
Preparing to unpack …/05-libjbig2dec0_0.13-6_amd64.deb …
Unpacking libjbig2dec0:amd64 (0.13-6) …
Selecting previously unselected package libgs9-common.
Preparing to unpack …/06-libgs9-common_9.26~dfsg+0-0ubuntu0.18.04.14_all.deb
…
Unpacking libgs9-common (9.26~dfsg+0-0ubuntu0.18.04.14) …
Selecting previously unselected package libgs9:amd64.
Preparing to unpack …/07-libgs9_9.26~dfsg+0-0ubuntu0.18.04.14_amd64.deb …
Unpacking libgs9:amd64 (9.26~dfsg+0-0ubuntu0.18.04.14) …
Selecting previously unselected package ghostscript.
Preparing to unpack …/08-ghostscript_9.26~dfsg+0-0ubuntu0.18.04.14_amd64.deb
…
Unpacking ghostscript (9.26~dfsg+0-0ubuntu0.18.04.14) …
Selecting previously unselected package gsfonts.
Preparing to unpack …/09-gsfonts_1%3a8.11+urwcyr1.0.7~pre44-4.4_all.deb …
Unpacking gsfonts (1:8.11+urwcyr1.0.7~pre44-4.4) …
Selecting previously unselected package libcupsfilters1:amd64.
Preparing to unpack …/10-libcupsfilters1_1.20.2-0ubuntu3.1_amd64.deb …
Unpacking libcupsfilters1:amd64 (1.20.2-0ubuntu3.1) …
Setting up libgs9-common (9.26~dfsg+0-0ubuntu0.18.04.14) …
Setting up fonts-droid-fallback (1:6.0.1r16-1.1) …
Setting up gsfonts (1:8.11+urwcyr1.0.7~pre44-4.4) …
Setting up poppler-data (0.4.8-2) …
Setting up fonts-noto-mono (20171026-2) …
Setting up libcupsfilters1:amd64 (1.20.2-0ubuntu3.1) …
Setting up libcupsimage2:amd64 (2.2.7-1ubuntu2.8) …
Setting up libjbig2dec0:amd64 (0.13-6) …
Setting up libijs-0.35:amd64 (0.35-13) …
Setting up libgs9:amd64 (9.26~dfsg+0-0ubuntu0.18.04.14) …
Setting up ghostscript (9.26~dfsg+0-0ubuntu0.18.04.14) …
Processing triggers for man-db (2.8.3-2ubuntu0.1) …
```

```
Processing triggers for fontconfig (2.12.6-0ubuntu2) …
Processing triggers for libc-bin (2.27-3ubuntu1.2) …
/sbin/ldconfig.real: /usr/local/lib/python3.7/dist-
packages/ideep4py/lib/libmkldnn.so.0 is not a symbolic link
```

0.1.1 EXERCISE-1: Build Grammar and Parser

[2]:
```
Grammar_1 = nltk.CFG.fromstring("""
S -> NP VP | NP VP
NP -> N | Det N | PRO | N N
VP -> V NP CP | VP ADVP | V NP
ADVP -> ADV ADV
CP -> COMP S
N -> 'Lisa' | 'brother' | 'peanut' | 'butter'
V -> 'told' | 'liked'
COMP -> 'that'
Det -> 'her'
PRO -> 'she'
ADV -> 'very' | 'much'

S -> NP VP
NP -> NP CONJ NP | N | NP PP | Det N | N | Det N
VP -> VP PP | VP CONJ VP | V | V
PP -> P NP | P NP
N -> 'Homer' | 'friends' | 'work' | 'bar'
V -> 'drank' | 'sang'
CONJ -> 'and' | 'and'
Det -> 'his' | 'the'
P -> 'from' | 'in'

S -> NP VP
NP -> NP CONJ NP | N | N
VP -> V ADJP
ADJP -> ADJP CONJ ADJP | ADJ | ADV ADJ
N -> 'Homer' | 'Marge'
V -> 'are'
CONJ -> 'and' | 'but'
ADJ -> 'poor' | 'happy'
ADV -> 'very'

S -> NP VP | NP AUX VP
NP -> PRO | NP CP | Det N | PRO | PRO | PRO | N |Det N
VP -> V NP PP | V NP NP
CP -> COMP S
PP -> P NP
Det -> 'the' | 'his'
```

```
PRO -> 'he' | 'I' | 'him'
N -> 'book' | 't' | 'sister'
V -> 'gave' | 'given'
COMP -> 'that'
AUX -> 'had'
P -> 'to'

S -> NP VP
NP -> PRO | Det N | Det N
VP -> V NP PP
PP -> P NP
Det -> 'the' | 'his'
PRO -> 'he'
N -> 'book' | 'sister'
V -> 'gave'
P -> 'to'

S -> NP VP
NP -> Det ADJ N | Det ADJ ADJ N | N
VP -> V NP|VP PP
PP -> P NP
Det -> 'the' | 'the'
ADJ -> 'big' | 'tiny' | 'nerdy'
N -> 'bully' | 'kid' | 'school'
V -> 'punched'
P -> 'after'
""")
```

1.Using NLTK's nltk.CFG.fromstring() method, build a CFG named grammar1. The grammar should cover all of the sentences below and their tree structure as presented on this page. The grammar's start symbol should be 'S': make sure that an S rule (ex. S -> NP VP) is the very top rule in your list of rules.

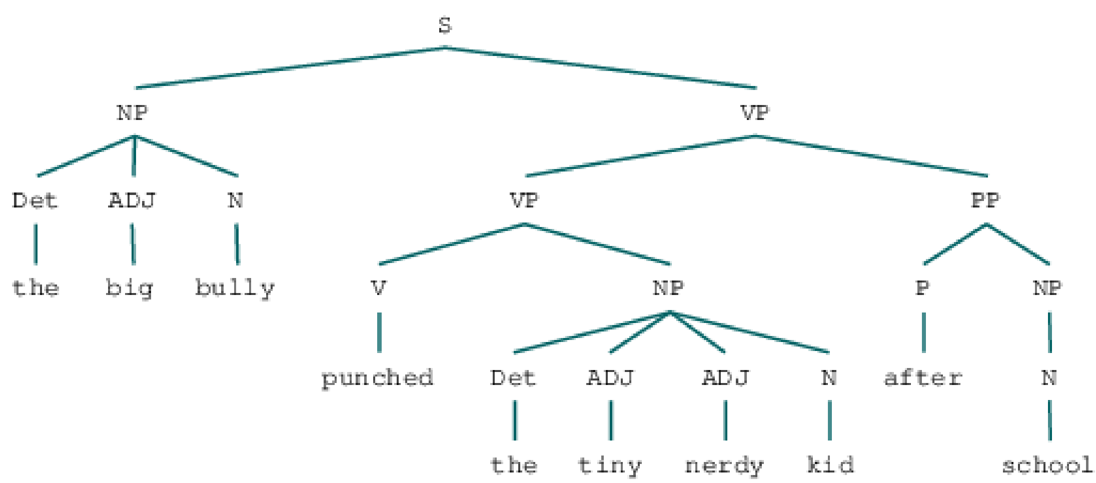
0.1.2 (s6)the big bully punched the tiny nerdy kid after school

[16]:
```
s6_grammar1 = nltk.CFG.fromstring("""
S -> NP VP
NP -> Det ADJ N | Det ADJ ADJ N | N
VP -> V NP|VP PP
PP -> P NP
Det -> 'the' | 'the'
ADJ -> 'big' | 'tiny' | 'nerdy'
N -> 'bully' | 'kid' | 'school'
V -> 'punched'
P -> 'after'
""")
```

[17]:
```python
sentence6 = word_tokenize("the big bully punched the tiny nerdy kid after school")
parser = nltk.ChartParser(s6_grammar1)
for tree in parser.parse(sentence6):
    print(tree)
```

```
(S
  (NP (Det the) (ADJ big) (N bully))
  (VP
    (VP (V punched) (NP (Det the) (ADJ tiny) (ADJ nerdy) (N kid)))
    (PP (P after) (NP (N school)))))
```

[18]:
```python
np6 =nltk.Tree.fromstring('(S(NP (Det the) (ADJ big) (N bully))(VP(VP (V punched) (NP (Det the) (ADJ tiny) (ADJ nerdy) (N kid)))(PP (P after) (NP (N school)))))')
display(np1)
```



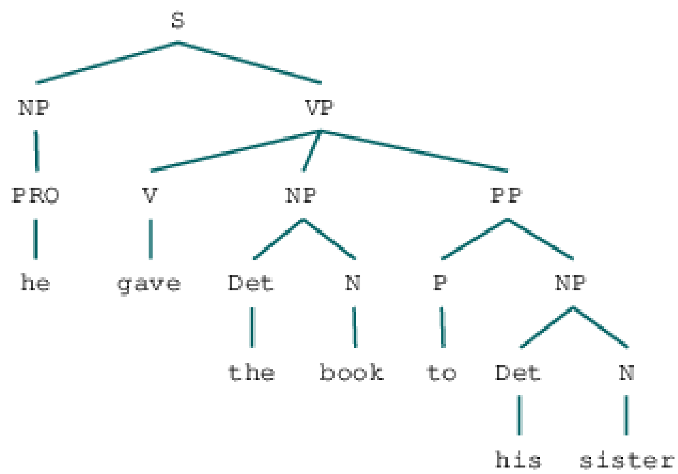
0.1.3 (s7)he gave the book to his sister

[19]:
```python
s7_grammar1 = nltk.CFG.fromstring("""
S -> NP VP
NP -> PRO | Det N | Det N
VP -> V NP PP
PP -> P NP
Det -> 'the' | 'his'
PRO -> 'he'
N -> 'book' | 'sister'
V -> 'gave'
P -> 'to'
""")
```

[20]:
```
sentence7 = word_tokenize("he gave the book to his sister")
parser = nltk.ChartParser(s7_grammar1)
for i in parser.parse(sentence7):
    print(i)
```

```
(S
  (NP (PRO he))
  (VP
    (V gave)
    (NP (Det the) (N book))
    (PP (P to) (NP (Det his) (N sister)))))
```

[21]:
```
np7 =nltk.Tree.fromstring('(S(NP (PRO he))(VP(V gave)(NP (Det the) (N book))(PP␣
 ↪(P to) (NP (Det his) (N sister)))))')
display(np7)
```



0.1.4 (s8)he gave the book that I had given him t to his sister

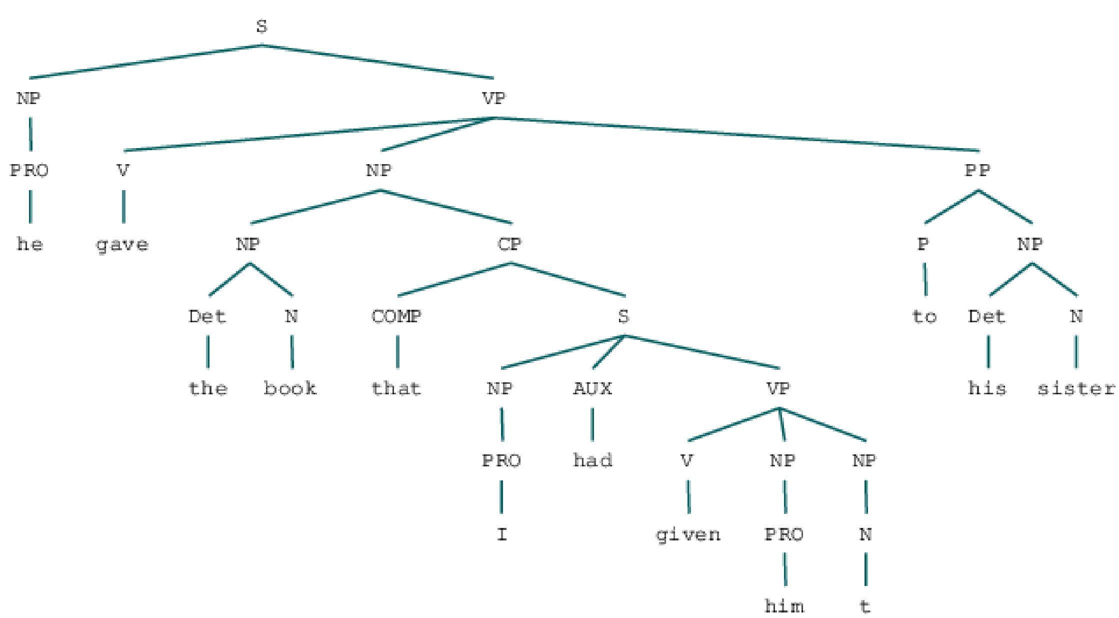
[22]:
```
s8_grammar1 = nltk.CFG.fromstring("""
S -> NP VP | NP AUX VP
NP -> PRO | NP CP | Det N | PRO | PRO | PRO | N |Det N
VP -> V NP PP | V NP NP
CP -> COMP S
PP -> P NP
Det -> 'the' | 'his'
PRO -> 'he' | 'I' | 'him'
N -> 'book' | 't' | 'sister'
V -> 'gave' | 'given'
COMP -> 'that'
AUX -> 'had'
```

```
P -> 'to'
""")
```

[23]:
```
sentence8 = word_tokenize("he gave the book that I had given him t to his␣
↪sister")
parser = nltk.ChartParser(s8_grammar1)
for i in parser.parse(sentence8):
    print(i)
```

```
(S
  (NP (PRO he))
  (VP
    (V gave)
    (NP
      (NP (Det the) (N book))
      (CP
        (COMP that)
        (S
          (NP (PRO I))
          (AUX had)
          (VP (V given) (NP (PRO him)) (NP (N t))))))
    (PP (P to) (NP (Det his) (N sister)))))
```

[24]:
```
np8 =nltk.Tree.fromstring('(S(NP (PRO he))(VP(V gave)(NP(NP (Det the) (N␣
↪book))(CP(COMP that)(S(NP (PRO I))(AUX had)(VP (V given) (NP (PRO him)) (NP␣
↪(N t))))))(PP (P to) (NP (Det his) (N sister)))))')
display(np8)
```



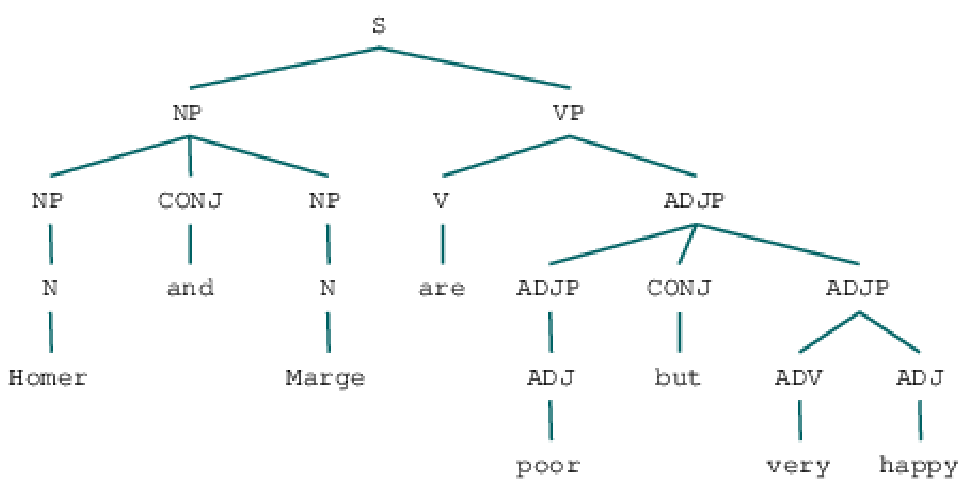
0.1.5 (s9)Homer and Marge are poor but very happy

[25]:
```
s9_grammar1 = nltk.CFG.fromstring("""
S -> NP VP
NP -> NP CONJ NP | N | N
VP -> V ADJP
ADJP -> ADJP CONJ ADJP | ADJ | ADV ADJ
N -> 'Homer' | 'Marge'
V -> 'are'
CONJ -> 'and' | 'but'
ADJ -> 'poor' | 'happy'
ADV -> 'very'
""")
```

[26]:
```
sentence9 = word_tokenize("Homer and Marge are poor but very happy")
parser = nltk.ChartParser(s9_grammar1)
for i in parser.parse(sentence9):
    print(i)
```

```
(S
  (NP (NP (N Homer)) (CONJ and) (NP (N Marge)))
  (VP
    (V are)
    (ADJP (ADJP (ADJ poor)) (CONJ but) (ADJP (ADV very) (ADJ happy)))))
```

[27]:
```
np9 =nltk.Tree.fromstring('(S(NP (NP (N Homer)) (CONJ and) (NP (N Marge)))(VP(V␣
↪are)(ADJP (ADJP (ADJ poor)) (CONJ but) (ADJP (ADV very) (ADJ happy)))))')
display(np9)
```



0.1.6 (s10)Homer and his friends from work drank and sang in the bar

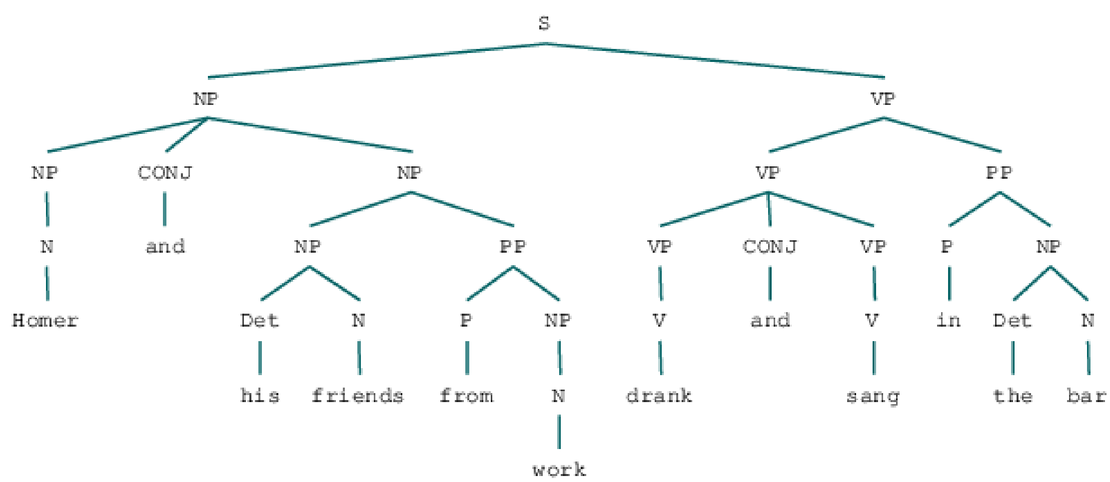
[28]:
```
s10_grammar1 = nltk.CFG.fromstring("""
S -> NP VP
NP -> NP CONJ NP | N | NP PP | Det N | N | Det N
VP -> VP PP | VP CONJ VP | V | V
PP -> P NP | P NP
N -> 'Homer' | 'friends' | 'work' | 'bar'
V -> 'drank' | 'sang'
CONJ -> 'and' | 'and'
Det -> 'his' | 'the'
P -> 'from' | 'in'
""")
```

[29]:
```
sentence10 = word_tokenize("Homer and his friends from work drank and sang in␣
↪the bar")
parser = nltk.ChartParser(s10_grammar1)
for i in parser.parse(sentence10):
    print(i)
```

```
(S
  (NP
    (NP (NP (N Homer)) (CONJ and) (NP (Det his) (N friends)))
    (PP (P from) (NP (N work))))
  (VP
    (VP (VP (V drank)) (CONJ and) (VP (V sang)))
    (PP (P in) (NP (Det the) (N bar)))))
(S
  (NP
    (NP (N Homer))
    (CONJ and)
    (NP (NP (Det his) (N friends)) (PP (P from) (NP (N work)))))
  (VP
    (VP (VP (V drank)) (CONJ and) (VP (V sang)))
    (PP (P in) (NP (Det the) (N bar)))))
(S
  (NP
    (NP (NP (N Homer)) (CONJ and) (NP (Det his) (N friends)))
    (PP (P from) (NP (N work))))
  (VP
    (VP (V drank))
    (CONJ and)
    (VP (VP (V sang)) (PP (P in) (NP (Det the) (N bar))))))
(S
  (NP
    (NP (N Homer))
    (CONJ and)
    (NP (NP (Det his) (N friends)) (PP (P from) (NP (N work)))))
```

```
  (VP
    (VP (V drank))
    (CONJ and)
    (VP (VP (V sang)) (PP (P in) (NP (Det the) (N bar))))))
```

[30]:
```python
np10 =nltk.Tree.fromstring('(S(NP(NP (N Homer))(CONJ and)(NP (NP (Det his) (N␣
↪friends)) (PP (P from) (NP (N work)))))(VP(VP (VP (V drank)) (CONJ and) (VP␣
↪(V sang)))(PP (P in) (NP (Det the) (N bar)))))')
display(np10)
```



0.1.7 (s11)Lisa told her brother that she liked peanut butter very much

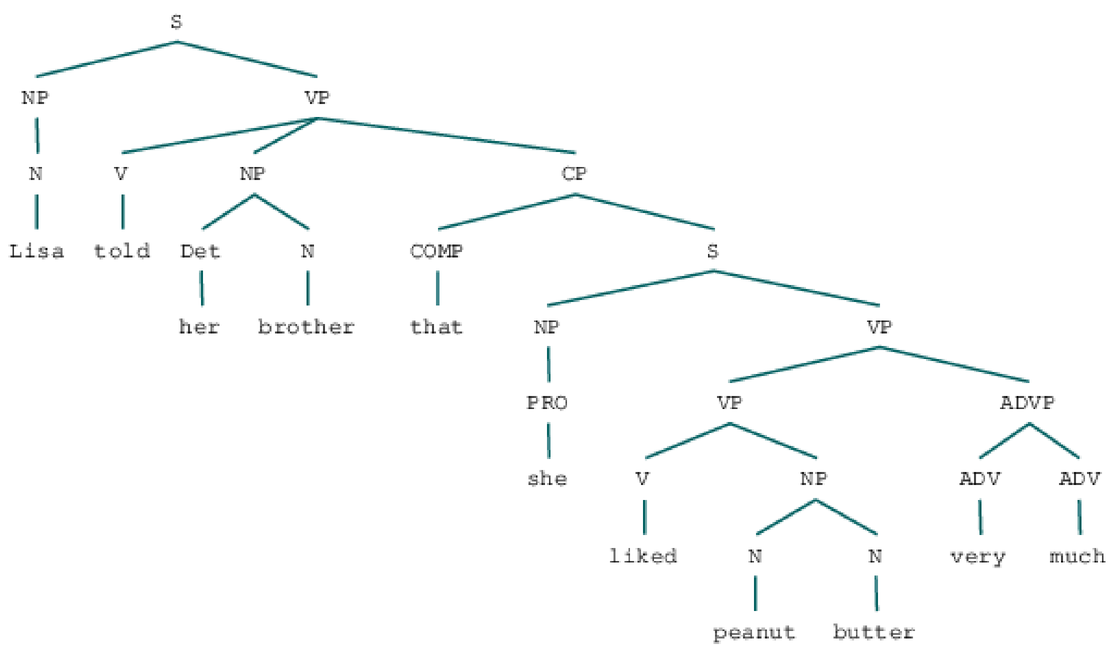
[31]:
```python
s11_grammar1 = nltk.CFG.fromstring("""
S -> NP VP | NP VP
NP -> N | Det N | PRO | N N
VP -> V NP CP | VP ADVP | V NP
ADVP -> ADV ADV
CP -> COMP S
N -> 'Lisa' | 'brother' | 'peanut' | 'butter'
V -> 'told' | 'liked'
COMP -> 'that'
Det -> 'her'
PRO -> 'she'
ADV -> 'very' | 'much'
""")
```

[32]:
```python
sentence11 = word_tokenize("Lisa told her brother that she liked peanut butter␣
↪very much")
parser = nltk.ChartParser(s11_grammar1)
for i in parser.parse(sentence11):
```

```
    print(i)
```

```
(S
  (NP (N Lisa))
  (VP
    (VP
      (V told)
      (NP (Det her) (N brother))
      (CP
        (COMP that)
        (S (NP (PRO she)) (VP (V liked) (NP (N peanut) (N butter))))))
    (ADVP (ADV very) (ADV much))))
(S
  (NP (N Lisa))
  (VP
    (V told)
    (NP (Det her) (N brother))
    (CP
      (COMP that)
      (S
        (NP (PRO she))
        (VP
          (VP (V liked) (NP (N peanut) (N butter)))
          (ADVP (ADV very) (ADV much)))))))
```

[33]:
```
np11 =nltk.Tree.fromstring('(S(NP (N Lisa))(VP(V told)(NP (Det her) (N␣
↪brother))(CP(COMP that)(S(NP (PRO she))(VP(VP (V liked) (NP (N peanut) (N␣
↪butter)))(ADVP (ADV very) (ADV much))))))))')
display(np11)
```



2.Once a grammar is built, you can print it. Also, you can extract a set of production rules with the .productions() method. Unlike the .productions() method called on a Tree object, the resulting list should be duplicate-free. As before, each rule in the list is a production rule type. A rule has a left-hand side node (the parent node), which you can getto using the .lhs() method; the actual string label for the node can be accessed by calling .symbol() on the node object.

[ ]:
```
grammer3 = nltk.CFG.fromstring("""
S -> NP VP
NP -> N
VP -> V
N -> 'Homer'
V -> 'sleeps'
""")
```

[ ]:
```
print(grammer3)
```

```
Grammar with 5 productions (start state = S)
    S -> NP VP
    NP -> N
    VP -> V
    N -> 'Homer'
    V -> 'sleeps'
```

[ ]:
```
grammer3.productions()
```

[ ]:
```
[S -> NP VP, NP -> N, VP -> V, N -> 'Homer', V -> 'sleeps']
```

[ ]:
```
last_rule = grammer3.productions()[-1]
```

[ ]:
```
last_rule
```

[ ]:
```
V -> 'sleeps'
```

[ ]:
```
last_rule.is_lexical()
```

[ ]:
```
True
```

[ ]:
```
last_rule.lhs()
```

[ ]:
```
V
```

[ ]:
```
last_rule.lhs().symbol()
```

[ ]:
```
'V'
```

0.2 3.Explore the rules and answer the following questions.

[34]:
```
Grammar_all = nltk.CFG.fromstring("""
S -> NP VP | NP AUX VP
NP -> Det ADJ N | N | PRO | Det N | PRO | NP CP | PRO | NP CONJ | NP PP | N N
VP -> V NP | VP PP | V NP PP | V NP | V ADJP | VP PP | VP CONJ | V NP CP | VP␣
↪ADVP
CP -> COMP S
PP -> P NP
Det -> 'the' | 'his' | 'her'
ADJ -> 'big' | 'tiny' | 'nerdy' | 'poor' | 'happy'
ADV -> 'very' | 'much'
PRO -> 'he' | 'I' | 'him' | 'she'
ADJP -> ADJP CONJ | ADJ
ADVP -> ADV
N -> 'bully' | 'kid' | 'school' | 'book' | 'sister' | 't' | 'Homer' | 'Marge'|␣
↪'friends' | 'work' | 'bar' | 'Lisa' | 'brother' | 'peanut' | 'butter'
V -> 'punched' | 'gave' | 'given' | 'are' | 'drank' | 'sang' | 'told' | 'liked'
CONJ -> 'and' | 'but'
COMP -> 'that'
AUX -> 'had'
P -> 'after' | 'to' | 'from' | 'in'
""")
```

a. What is the start state of your grammar?

[35]:
```
Grammar_all.productions()[0].lhs()
```

[35]: S

b. How many CF rules are in your grammar?

[36]:
```
len(Grammar_all.productions())
```

[36]: 71

c. How many of them are lexical?

[37]:
```
n=0
for x in Grammar_all.productions():
    if x.is_lexical():
        n = n+1
print("How many of them are lexical?  ",n)
```

```
How many of them are lexical?   45
```

d. How many VP rules are there? That is, how many rules have 'VP' on the left-hand side of the rule? That is, how many rules are of the VP -> ... form?

[38]:
```
n=0
for x in Grammar_all.productions():
```

```
    if x.lhs().symbol() == 'VP':
        n = n+1
n
```

[38]: 9

e. How many V rules are there? That is, how many rules have 'V' on the left-hand side of the fule? That is, how many rules are of the V -> ... form?

[39]:
```
n=0
for x in Grammar_all.productions():
    if x.lhs().symbol() == 'V':
        n = n+1
n
```

[39]: 8

0.2.1 4.Using grammar1, build a chart parser.

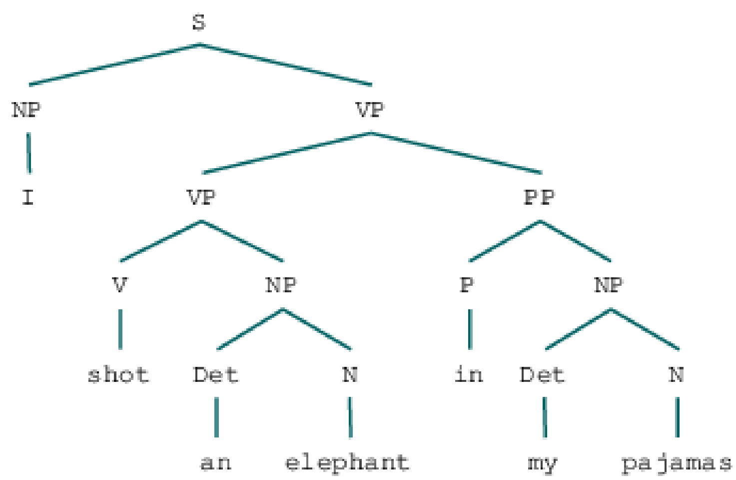
[41]:
```
sentence = word_tokenize("Lisa told her brother that she liked peanut butter␣
 ↪very much")
parser = nltk.ChartParser(Grammar_all)
for i in parser.parse(sentence):
    print(i)
```

```
(S
  (NP (N Lisa))
  (VP
    (V told)
    (NP (Det her) (N brother))
    (CP
      (COMP that)
      (S
        (NP (PRO she))
        (VP
          (VP
            (VP (V liked) (NP (N peanut) (N butter)))
            (ADVP (ADV very)))
          (ADVP (ADV much)))))))
(S
  (NP (N Lisa))
  (VP
    (V told)
    (NP
      (NP (Det her) (N brother))
      (CP
        (COMP that)
        (S
          (NP (PRO she))
```

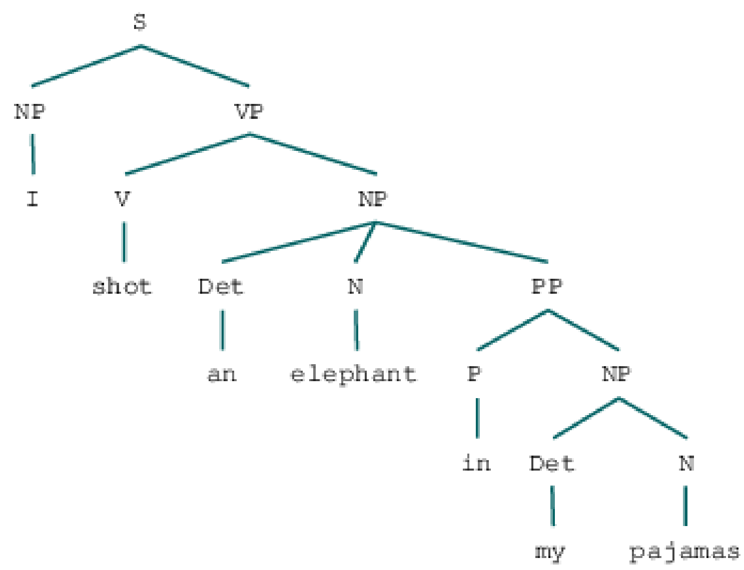
```
          (VP
            (VP
              (VP (V liked) (NP (N peanut) (N butter)))
              (ADVP (ADV very)))
            (ADVP (ADV much)))))))
(S
  (NP (N Lisa))
  (VP
    (VP
      (VP
        (V told)
        (NP (Det her) (N brother))
        (CP
          (COMP that)
          (S
            (NP (PRO she))
            (VP (V liked) (NP (N peanut) (N butter))))))
      (ADVP (ADV very)))
    (ADVP (ADV much))))
(S
  (NP (N Lisa))
  (VP
    (VP
      (VP
        (V told)
        (NP
          (NP (Det her) (N brother))
          (CP
            (COMP that)
            (S
              (NP (PRO she))
              (VP (V liked) (NP (N peanut) (N butter)))))))
      (ADVP (ADV very)))
    (ADVP (ADV much))))
(S
  (NP (N Lisa))
  (VP
    (VP
      (V told)
      (NP (Det her) (N brother))
      (CP
        (COMP that)
        (S
          (NP (PRO she))
          (VP
            (VP (V liked) (NP (N peanut) (N butter)))
            (ADVP (ADV very))))))
    (ADVP (ADV much))))
```

```
(S
  (NP (N Lisa))
  (VP
    (VP
      (V told)
      (NP
        (NP (Det her) (N brother))
        (CP
          (COMP that)
          (S
            (NP (PRO she))
            (VP
              (VP (V liked) (NP (N peanut) (N butter)))
              (ADVP (ADV very)))))))
    (ADVP (ADV much))))
```

[42]:
```
q41 =nltk.Tree.fromstring('(S (NP I) (VP (VP (V shot) (NP (Det an) (N␣
↪elephant))) (PP (P in) (NP (Det my) (N pajamas)))))')
display(q41)
```



[43]:
```
q42 =nltk.Tree.fromstring('(S (NP I) (VP (V shot) (NP (Det an) (N elephant) (PP␣
↪(P in) (NP (Det my) (N pajamas))))))')
display(q42)
```



0.2.2 5. Using the parser, parse the sentences s6 – s11. If your grammar1 is built correctly to cover all of the sentences, the parser should successfully parse all of them.

[44]:
```
!pip install simple-colors
from simple_colors import *
```

```
Collecting simple-colors
  Downloading https://files.pythonhosted.org/packages/0f/07/e6710827a51f6bb5ef67
1f84db98275b122ae3e382726041c823bead1a84/simple_colors-0.1.5-py3-none-any.whl
Installing collected packages: simple-colors
Successfully installed simple-colors-0.1.5
```

[45]:
```
print(black("(s6):the big bully punched the tiny nerdy kid after␣
 ↪school","bold"))
print("\n")
sent6 = word_tokenize("the big bully punched the tiny nerdy kid after school")
parser = nltk.ChartParser(Grammar_1)
for i in parser.parse(sent6):
    print(i)
print("-------------------------------------------------------------------------------")
print("\n")
print(black("(s7):he gave the book to his sister","bold"))
print("\n")
sent7 = word_tokenize("he gave the book to his sister")
parser = nltk.ChartParser(Grammar_1)
for i in parser.parse(sent7):
    print(i)
```

```
print("-------------------------------------------------------------------------------")
print("\n")
print(black("(s8):he gave the book that I had given him t to his␣
↪sister","bold"))
print("\n")
sent8 = word_tokenize("he gave the book that I had given him t to his sister")
parser = nltk.ChartParser(Grammar_1)
for i in parser.parse(sent8):
    print(i)
print("-------------------------------------------------------------------------------")
print("\n")
print(black("(s9):Homer and Marge are poor but very happy","bold"))
print("\n")
sent9 = word_tokenize("Homer and Marge are poor but very happy")
parser = nltk.ChartParser(Grammar_1)
for i in parser.parse(sent9):
    print(i)
print("-------------------------------------------------------------------------------")
print("\n")
print(black("(s10):Homer and his friends from work drank and sang in the␣
↪bar","bold"))
print("\n")
sent10 = word_tokenize("Homer and his friends from work drank and sang in the␣
↪bar")
parser = nltk.ChartParser(Grammar_1)
for i in parser.parse(sent10):
    print(i)
print("-------------------------------------------------------------------------------")
print("\n")
print(black("(s11):Lisa told her brother that she liked peanut butter very␣
↪much","bold"))
print("\n")
sent11 = word_tokenize("Lisa told her brother that she liked peanut butter very␣
↪much")
parser = nltk.ChartParser(Grammar_1)
for i in parser.parse(sent11):
    print(i)
```

```
(s6):the big bully punched the tiny nerdy kid after school


(S
  (NP (Det the) (ADJ big) (N bully))
  (VP
    (VP (V punched) (NP (Det the) (ADJ tiny) (ADJ nerdy) (N kid)))
    (PP (P after) (NP (N school)))))
(S
```

```
  (NP (Det the) (ADJ big) (N bully))
  (VP
    (V punched)
    (NP (Det the) (ADJ tiny) (ADJ nerdy) (N kid))
    (PP (P after) (NP (N school)))))
(S
  (NP (Det the) (ADJ big) (N bully))
  (VP
    (V punched)
    (NP
      (NP (Det the) (ADJ tiny) (ADJ nerdy) (N kid))
      (PP (P after) (NP (N school))))))
-------------------------------------------------------------------------------
```

(s7):he gave the book to his sister

```
(S
  (NP (PRO he))
  (VP
    (VP (V gave) (NP (Det the) (N book)))
    (PP (P to) (NP (Det his) (N sister)))))
(S
  (NP (PRO he))
  (VP
    (V gave)
    (NP (Det the) (N book))
    (PP (P to) (NP (Det his) (N sister)))))
(S
  (NP (PRO he))
  (VP
    (V gave)
    (NP
      (NP (Det the) (N book))
      (PP (P to) (NP (Det his) (N sister))))))
-------------------------------------------------------------------------------
```

(s8):he gave the book that I had given him t to his sister

```
(S
  (NP (PRO he))
  (VP
    (V gave)
    (NP
      (NP (Det the) (N book))
```

```
      (CP
        (COMP that)
        (S
          (NP (PRO I))
          (AUX had)
          (VP (V given) (NP (PRO him)) (NP (N t))))))
    (PP (P to) (NP (Det his) (N sister)))))
(S
  (NP (PRO he))
  (VP
    (V gave)
    (NP
      (NP
        (NP (Det the) (N book))
        (CP
          (COMP that)
          (S
            (NP (PRO I))
            (AUX had)
            (VP (V given) (NP (PRO him)) (NP (N t))))))
      (PP (P to) (NP (Det his) (N sister))))))
(S
  (NP (PRO he))
  (VP
    (V gave)
    (NP
      (NP (Det the) (N book))
      (CP
        (COMP that)
        (S
          (NP (PRO I))
          (AUX had)
          (VP
            (VP (V given) (NP (PRO him)) (NP (N t)))
            (PP (P to) (NP (Det his) (N sister))))))))))
(S
  (NP (PRO he))
  (VP
    (V gave)
    (NP
      (NP (Det the) (N book))
      (CP
        (COMP that)
        (S
          (NP (PRO I))
          (AUX had)
          (VP
            (V given)
```

```
          (NP (PRO him))
          (NP (NP (N t)) (PP (P to) (NP (Det his) (N sister)))))))))
(S
  (NP (PRO he))
  (VP
    (V gave)
    (NP (Det the) (N book))
    (CP
      (COMP that)
      (S
        (NP (PRO I))
        (AUX had)
        (VP
          (VP (V given) (NP (PRO him)) (NP (N t)))
          (PP (P to) (NP (Det his) (N sister))))))))
(S
  (NP (PRO he))
  (VP
    (V gave)
    (NP (Det the) (N book))
    (CP
      (COMP that)
      (S
        (NP (PRO I))
        (AUX had)
        (VP
          (V given)
          (NP (PRO him))
          (NP (NP (N t)) (PP (P to) (NP (Det his) (N sister))))))))))
(S
  (NP (PRO he))
  (VP
    (VP
      (V gave)
      (NP
        (NP (Det the) (N book))
        (CP
          (COMP that)
          (S (NP (PRO I)) (AUX had) (VP (V given) (NP (PRO him))))))
      (NP (N t)))
    (PP (P to) (NP (Det his) (N sister)))))
(S
  (NP (PRO he))
  (VP
    (VP
      (V gave)
      (NP (Det the) (N book))
      (CP
```

```
        (COMP that)
        (S
          (NP (PRO I))
          (AUX had)
          (VP (V given) (NP (PRO him)) (NP (N t))))))
    (PP (P to) (NP (Det his) (N sister)))))
(S
  (NP (PRO he))
  (VP
    (VP
      (V gave)
      (NP
        (NP (Det the) (N book))
        (CP
          (COMP that)
          (S
            (NP (PRO I))
            (AUX had)
            (VP (V given) (NP (PRO him)) (NP (N t)))))))
    (PP (P to) (NP (Det his) (N sister)))))
(S
  (NP (PRO he))
  (VP
    (V gave)
    (NP
      (NP (Det the) (N book))
      (CP
        (COMP that)
        (S (NP (PRO I)) (AUX had) (VP (V given) (NP (PRO him))))))
    (NP (NP (N t)) (PP (P to) (NP (Det his) (N sister))))))
-------------------------------------------------------------------------------
```

(s9):Homer and Marge are poor but very happy

```
(S
  (NP (NP (N Homer)) (CONJ and) (NP (N Marge)))
  (VP
    (V are)
    (ADJP (ADJP (ADJ poor)) (CONJ but) (ADJP (ADV very) (ADJ happy)))))
-------------------------------------------------------------------------------
```

(s10):Homer and his friends from work drank and sang in the bar

```
(S
```

```
  (NP
    (NP (NP (N Homer)) (CONJ and) (NP (Det his) (N friends)))
    (PP (P from) (NP (N work))))
  (VP
    (VP (VP (V drank)) (CONJ and) (VP (V sang)))
    (PP (P in) (NP (Det the) (N bar)))))
(S
  (NP
    (NP (N Homer))
    (CONJ and)
    (NP (NP (Det his) (N friends)) (PP (P from) (NP (N work)))))
  (VP
    (VP (VP (V drank)) (CONJ and) (VP (V sang)))
    (PP (P in) (NP (Det the) (N bar)))))
(S
  (NP
    (NP (NP (N Homer)) (CONJ and) (NP (Det his) (N friends)))
    (PP (P from) (NP (N work))))
  (VP
    (VP (V drank))
    (CONJ and)
    (VP (VP (V sang)) (PP (P in) (NP (Det the) (N bar))))))
(S
  (NP
    (NP (N Homer))
    (CONJ and)
    (NP (NP (Det his) (N friends)) (PP (P from) (NP (N work)))))
  (VP
    (VP (V drank))
    (CONJ and)
    (VP (VP (V sang)) (PP (P in) (NP (Det the) (N bar))))))
--------------------------------------------------------------------------------
```

(s11):Lisa told her brother that she liked peanut butter very much

```
(S
  (NP (N Lisa))
  (VP
    (V told)
    (NP (Det her) (N brother))
    (CP
      (COMP that)
      (S
        (NP (PRO she))
        (VP
          (VP (V liked) (NP (N peanut) (N butter)))
```

```
          (ADVP (ADV very) (ADV much))))))
(S
  (NP (N Lisa))
  (VP
    (V told)
    (NP (Det her) (N brother))
    (CP
      (COMP that)
      (S
        (NP (PRO she))
        (VP
          (VP (V liked) (NP (N peanut)) (NP (N butter)))
          (ADVP (ADV very) (ADV much))))))
(S
  (NP (N Lisa))
  (VP
    (V told)
    (NP
      (NP (Det her) (N brother))
      (CP
        (COMP that)
        (S
          (NP (PRO she))
          (VP
            (VP (V liked) (NP (N peanut) (N butter)))
            (ADVP (ADV very) (ADV much)))))))
(S
  (NP (N Lisa))
  (VP
    (V told)
    (NP
      (NP (Det her) (N brother))
      (CP
        (COMP that)
        (S
          (NP (PRO she))
          (VP
            (VP (V liked) (NP (N peanut)) (NP (N butter)))
            (ADVP (ADV very) (ADV much)))))))
(S
  (NP (N Lisa))
  (VP
    (VP
      (V told)
      (NP
        (NP (Det her) (N brother))
        (CP (COMP that) (S (NP (PRO she)) (VP (V liked)))))
      (NP (N peanut) (N butter)))
```

```
    (ADVP (ADV very) (ADV much))))
(S
  (NP (N Lisa))
  (VP
    (VP
      (V told)
      (NP
        (NP (Det her) (N brother))
        (CP
          (COMP that)
          (S (NP (PRO she)) (VP (V liked) (NP (N peanut))))))
      (NP (N butter)))
    (ADVP (ADV very) (ADV much))))
(S
  (NP (N Lisa))
  (VP
    (VP
      (V told)
      (NP (Det her) (N brother))
      (CP
        (COMP that)
        (S (NP (PRO she)) (VP (V liked) (NP (N peanut) (N butter))))))
    (ADVP (ADV very) (ADV much))))
(S
  (NP (N Lisa))
  (VP
    (VP
      (V told)
      (NP (Det her) (N brother))
      (CP
        (COMP that)
        (S
          (NP (PRO she))
          (VP (V liked) (NP (N peanut)) (NP (N butter))))))
    (ADVP (ADV very) (ADV much))))
(S
  (NP (N Lisa))
  (VP
    (VP
      (V told)
      (NP
        (NP (Det her) (N brother))
        (CP
          (COMP that)
          (S
            (NP (PRO she))
            (VP (V liked) (NP (N peanut) (N butter)))))))
    (ADVP (ADV very) (ADV much))))
```

```
(S
  (NP (N Lisa))
  (VP
    (VP
      (V told)
      (NP
        (NP (Det her) (N brother))
        (CP
          (COMP that)
          (S
            (NP (PRO she))
            (VP (V liked) (NP (N peanut)) (NP (N butter)))))))
    (ADVP (ADV very) (ADV much))))
```

```
[ ]:
```

lab13_nlp_viviyan_33

June 8, 2021

0.0.1 viviyan richards w

205229133

0.1 Lab13. Improving Grammar to Parse Ambiguous Sentences

[1]:
```
import nltk
from nltk.tree import Tree
from nltk.tokenize import word_tokenize
from IPython.display import display
import nltk,re,pprint
from nltk.tag import pos_tag
from nltk.chunk import ne_chunk
import numpy as npt
```

In this lab, you will refine the grammar you have built in the previous lab. Because, the grammar does not parse some sentences that are ambiguous. #### EXERCISE-1

[2]:
```
Grammar_1 = nltk.CFG.fromstring("""
S -> NP VP | NP VP
NP -> N | Det N | PRO | N N
VP -> V NP CP | VP ADVP | V NP
ADVP -> ADV ADV
CP -> COMP S
N -> 'Lisa' | 'brother' | 'peanut' | 'butter'
V -> 'told' | 'liked'
COMP -> 'that'
Det -> 'her'
PRO -> 'she'
ADV -> 'very' | 'much'

S -> NP VP
NP -> NP CONJ NP | N | NP PP | Det N | N | Det N
VP -> VP PP | VP CONJ VP | V | V
PP -> P NP | P NP
N -> 'Homer' | 'friends' | 'work' | 'bar'
V -> 'drank' | 'sang'
CONJ -> 'and' | 'and'
Det -> 'his' | 'the'
```

```
P -> 'from' | 'in'

S -> NP VP
NP -> NP CONJ NP | N | N
VP -> V ADJP
ADJP -> ADJP CONJ ADJP | ADJ | ADV ADJ
N -> 'Homer' | 'Marge'
V -> 'are'
CONJ -> 'and' | 'but'
ADJ -> 'poor' | 'happy'
ADV -> 'very'

S -> NP VP | NP AUX VP
NP -> PRO | NP CP | Det N | PRO | PRO | PRO | N |Det N
VP -> V NP PP | V NP NP
CP -> COMP S
PP -> P NP
Det -> 'the' | 'his'
PRO -> 'he' | 'I' | 'him'
N -> 'book' | 't' | 'sister'
V -> 'gave' | 'given'
COMP -> 'that'
AUX -> 'had'
P -> 'to'

S -> NP VP
NP -> PRO | Det N | Det N
VP -> V NP PP
PP -> P NP
Det -> 'the' | 'his'
PRO -> 'he'
N -> 'book' | 'sister'
V -> 'gave'
P -> 'to'

S -> NP VP
NP -> Det ADJ N | Det ADJ ADJ N | N
VP -> V NP|VP PP
PP -> P NP
Det -> 'the' | 'the'
ADJ -> 'big' | 'tiny' | 'nerdy'
N -> 'bully' | 'kid' | 'school'
V -> 'punched'
P -> 'after'
""")
```

In this part, you will be updating the grammar and the parser you built in the previous lab. ### 1. Examine the parser output from the previous lab. Is any of the sentences ambiguous, that is, has more than one parse tree? Pick an example and provide an explanation.

Yes! here we have two sentences which has more than one parse tree

1. Homer and his friends from work drank and sang in the bar
2. Lisa told her brother that she liked peanut butter very much

[10]:
```
sentence5 = word_tokenize("Homer and his friends from work drank and sang in␣
↪the bar")
par = nltk.ChartParser(Grammar_1)
for i in par.parse(sentence5):
    print(i)
```

```
(S
  (NP
    (NP (NP (N Homer)) (CONJ and) (NP (Det his) (N friends)))
    (PP (P from) (NP (N work))))
  (VP
    (VP (VP (V drank)) (CONJ and) (VP (V sang)))
    (PP (P in) (NP (Det the) (N bar)))))
(S
  (NP
    (NP (N Homer))
    (CONJ and)
    (NP (NP (Det his) (N friends)) (PP (P from) (NP (N work)))))
  (VP
    (VP (VP (V drank)) (CONJ and) (VP (V sang)))
    (PP (P in) (NP (Det the) (N bar)))))
(S
  (NP
    (NP (NP (N Homer)) (CONJ and) (NP (Det his) (N friends)))
    (PP (P from) (NP (N work))))
  (VP
    (VP (V drank))
    (CONJ and)
    (VP (VP (V sang)) (PP (P in) (NP (Det the) (N bar))))))
(S
  (NP
    (NP (N Homer))
    (CONJ and)
    (NP (NP (Det his) (N friends)) (PP (P from) (NP (N work)))))
  (VP
    (VP (V drank))
    (CONJ and)
    (VP (VP (V sang)) (PP (P in) (NP (Det the) (N bar))))))
```

[11]:
```
sentence6 = word_tokenize("Lisa told her brother that she liked peanut butter␣
↪very much")
```

```python
par = nltk.ChartParser(Grammar_1)
for i in par.parse(sentence6):
    print(i)
```

```
(S
  (NP (N Lisa))
  (VP
    (V told)
    (NP (Det her) (N brother))
    (CP
      (COMP that)
      (S
        (NP (PRO she))
        (VP
          (VP (V liked) (NP (N peanut) (N butter)))
          (ADVP (ADV very) (ADV much)))))))
(S
  (NP (N Lisa))
  (VP
    (V told)
    (NP (Det her) (N brother))
    (CP
      (COMP that)
      (S
        (NP (PRO she))
        (VP
          (VP (V liked) (NP (N peanut)) (NP (N butter)))
          (ADVP (ADV very) (ADV much)))))))
(S
  (NP (N Lisa))
  (VP
    (V told)
    (NP
      (NP (Det her) (N brother))
      (CP
        (COMP that)
        (S
          (NP (PRO she))
          (VP
            (VP (V liked) (NP (N peanut) (N butter)))
            (ADVP (ADV very) (ADV much))))))))
(S
  (NP (N Lisa))
  (VP
    (V told)
    (NP
      (NP (Det her) (N brother))
      (CP
```

```
        (COMP that)
        (S
          (NP (PRO she))
          (VP
            (VP (V liked) (NP (N peanut)) (NP (N butter)))
            (ADVP (ADV very) (ADV much))))))))
(S
  (NP (N Lisa))
  (VP
    (VP
      (V told)
      (NP
        (NP (Det her) (N brother))
        (CP (COMP that) (S (NP (PRO she)) (VP (V liked)))))
      (NP (N peanut) (N butter)))
    (ADVP (ADV very) (ADV much))))
(S
  (NP (N Lisa))
  (VP
    (VP
      (V told)
      (NP
        (NP (Det her) (N brother))
        (CP
          (COMP that)
          (S (NP (PRO she)) (VP (V liked) (NP (N peanut))))))
      (NP (N butter)))
    (ADVP (ADV very) (ADV much))))
(S
  (NP (N Lisa))
  (VP
    (VP
      (V told)
      (NP (Det her) (N brother))
      (CP
        (COMP that)
        (S (NP (PRO she)) (VP (V liked) (NP (N peanut) (N butter))))))
    (ADVP (ADV very) (ADV much))))
(S
  (NP (N Lisa))
  (VP
    (VP
      (V told)
      (NP (Det her) (N brother))
      (CP
        (COMP that)
        (S
          (NP (PRO she))
```

```
        (VP (V liked) (NP (N peanut)) (NP (N butter))))))
    (ADVP (ADV very) (ADV much))))
(S
  (NP (N Lisa))
  (VP
    (VP
      (V told)
      (NP
        (NP (Det her) (N brother))
        (CP
          (COMP that)
          (S
            (NP (PRO she))
            (VP (V liked) (NP (N peanut) (N butter)))))))
    (ADVP (ADV very) (ADV much))))
(S
  (NP (N Lisa))
  (VP
    (VP
      (V told)
      (NP
        (NP (Det her) (N brother))
        (CP
          (COMP that)
          (S
            (NP (PRO she))
            (VP (V liked) (NP (N peanut)) (NP (N butter)))))))
    (ADVP (ADV very) (ADV much))))
```

0.1.1 2. Have your parser parse this new sentence. It is covered by the grammar, therefore the

parser should be able to handle it:

(s12): Lisa and her friends told Marge that Homer punched the bully in the bar

[12]:
```
s12 = word_tokenize("Lisa and her friends told Marge that Homer punched the␣
↪bully in the bar")
par = nltk.ChartParser(Grammar_1)
for i in par.parse(s12):
    print(i)
```

```
(S
  (NP (NP (N Lisa)) (CONJ and) (NP (Det her) (N friends)))
  (VP
    (V told)
    (NP
      (NP (N Marge))
      (CP
```

```
        (COMP that)
        (S (NP (N Homer)) (VP (V punched) (NP (Det the) (N bully))))))
    (PP (P in) (NP (Det the) (N bar)))))
(S
  (NP (NP (N Lisa)) (CONJ and) (NP (Det her) (N friends)))
  (VP
    (V told)
    (NP
      (NP
        (NP (N Marge))
        (CP
          (COMP that)
          (S
            (NP (N Homer))
            (VP (V punched) (NP (Det the) (N bully))))))
      (PP (P in) (NP (Det the) (N bar))))))
(S
  (NP (NP (N Lisa)) (CONJ and) (NP (Det her) (N friends)))
  (VP
    (V told)
    (NP
      (NP (N Marge))
      (CP
        (COMP that)
        (S
          (NP (N Homer))
          (VP
            (VP (V punched) (NP (Det the) (N bully)))
            (PP (P in) (NP (Det the) (N bar)))))))))
(S
  (NP (NP (N Lisa)) (CONJ and) (NP (Det her) (N friends)))
  (VP
    (V told)
    (NP
      (NP (N Marge))
      (CP
        (COMP that)
        (S
          (NP (N Homer))
          (VP
            (V punched)
            (NP (Det the) (N bully))
            (PP (P in) (NP (Det the) (N bar)))))))))
(S
  (NP (NP (N Lisa)) (CONJ and) (NP (Det her) (N friends)))
  (VP
    (V told)
    (NP
```

```
      (NP (N Marge))
      (CP
        (COMP that)
        (S
          (NP (N Homer))
          (VP
            (V punched)
            (NP
              (NP (Det the) (N bully))
              (PP (P in) (NP (Det the) (N bar))))))))))
(S
  (NP (NP (N Lisa)) (CONJ and) (NP (Det her) (N friends)))
  (VP
    (V told)
    (NP (N Marge))
    (CP
      (COMP that)
      (S
        (NP (N Homer))
        (VP
          (VP (V punched) (NP (Det the) (N bully)))
          (PP (P in) (NP (Det the) (N bar))))))))
(S
  (NP (NP (N Lisa)) (CONJ and) (NP (Det her) (N friends)))
  (VP
    (V told)
    (NP (N Marge))
    (CP
      (COMP that)
      (S
        (NP (N Homer))
        (VP
          (V punched)
          (NP (Det the) (N bully))
          (PP (P in) (NP (Det the) (N bar))))))))
(S
  (NP (NP (N Lisa)) (CONJ and) (NP (Det her) (N friends)))
  (VP
    (V told)
    (NP (N Marge))
    (CP
      (COMP that)
      (S
        (NP (N Homer))
        (VP
          (V punched)
          (NP
            (NP (Det the) (N bully))
```

```
            (PP (P in) (NP (Det the) (N bar))))))))
(S
  (NP (NP (N Lisa)) (CONJ and) (NP (Det her) (N friends)))
  (VP
    (VP
      (V told)
      (NP
        (NP (N Marge))
        (CP (COMP that) (S (NP (N Homer)) (VP (V punched)))))
      (NP (Det the) (N bully)))
    (PP (P in) (NP (Det the) (N bar)))))
(S
  (NP (NP (N Lisa)) (CONJ and) (NP (Det her) (N friends)))
  (VP
    (VP
      (V told)
      (NP (N Marge))
      (CP
        (COMP that)
        (S (NP (N Homer)) (VP (V punched) (NP (Det the) (N bully))))))
    (PP (P in) (NP (Det the) (N bar)))))
(S
  (NP (NP (N Lisa)) (CONJ and) (NP (Det her) (N friends)))
  (VP
    (VP
      (V told)
      (NP
        (NP (N Marge))
        (CP
          (COMP that)
          (S
            (NP (N Homer))
            (VP (V punched) (NP (Det the) (N bully)))))))
    (PP (P in) (NP (Det the) (N bar)))))
(S
  (NP (NP (N Lisa)) (CONJ and) (NP (Det her) (N friends)))
  (VP
    (V told)
    (NP
      (NP (N Marge))
      (CP (COMP that) (S (NP (N Homer)) (VP (V punched)))))
    (NP (NP (Det the) (N bully)) (PP (P in) (NP (Det the) (N bar))))))
```

0.1.2 3. Come up with a sentence of your own that's covered by grammar1 and have the parser parse it. Are you satisfied with the result?

```
[13]: result = word_tokenize("Homer and friends punched the tiny nerdy kid after␣
      ↪school")
      par = nltk.ChartParser(Grammar_1)
      for i in par.parse(result):
          print(i)
```

```
(S
  (NP (NP (N Homer)) (CONJ and) (NP (N friends)))
  (VP
    (VP (V punched) (NP (Det the) (ADJ tiny) (ADJ nerdy) (N kid)))
    (PP (P after) (NP (N school)))))
(S
  (NP (NP (N Homer)) (CONJ and) (NP (N friends)))
  (VP
    (V punched)
    (NP (Det the) (ADJ tiny) (ADJ nerdy) (N kid))
    (PP (P after) (NP (N school)))))
(S
  (NP (NP (N Homer)) (CONJ and) (NP (N friends)))
  (VP
    (V punched)
    (NP
      (NP (Det the) (ADJ tiny) (ADJ nerdy) (N kid))
      (PP (P after) (NP (N school))))))
```

0.1.3 4. Let's revisit our first three sentences from the previous lab.

(s1): Marge will make a ham sandwich

(s2): will Marge make a ham sandwich

(s3): Homer ate the donut on the table

As it is, your grammar1 does not cover them. But we can extend it with the CF rules from the three sentences' trees. Follow the steps below.

a. From the three sentence trees, create a list of all production rules in them. Turn it into a set, which removes all duplicates. (Hint: use set().)

```
[14]: a_set = set()
```

```
[15]: s1 = Tree.fromstring('(S(NP (N Marge))(AUX will)(VP (V make) (NP (Det a) (N␣
      ↪ham) (N sandwich))))')
      s1_r = s1.productions()
      s1_r
```

[15]:
```
[S -> NP AUX VP,
 NP -> N,
 N -> 'Marge',
 AUX -> 'will',
 VP -> V NP,
 V -> 'make',
 NP -> Det N N,
 Det -> 'a',
 N -> 'ham',
 N -> 'sandwich']
```

[16]:
```
s2 = Tree.fromstring('(S(AUX will)(NP (N Marge))(VP (V make) (NP (Det a) (N␣
↪ham) (N sandwich))))')
s2_r = s2.productions()
s2_r
```

[16]:
```
[S -> AUX NP VP,
 AUX -> 'will',
 NP -> N,
 N -> 'Marge',
 VP -> V NP,
 V -> 'make',
 NP -> Det N N,
 Det -> 'a',
 N -> 'ham',
 N -> 'sandwich']
```

[17]:
```
s3 = Tree.fromstring('(S(NP (N Homer))(VP(V ate)(NP(NP (Det the) (N donut))(PP␣
↪(P on) (NP (Det the) (N table))))))')
s3_r = s3.productions()
s3_r
```

[17]:
```
[S -> NP VP,
 NP -> N,
 N -> 'Homer',
 VP -> V NP,
 V -> 'ate',
 NP -> NP PP,
 NP -> Det N,
 Det -> 'the',
 N -> 'donut',
 PP -> P NP,
 P -> 'on',
 NP -> Det N,
 Det -> 'the',
 N -> 'table']
```

[18]:
```
s_1r = []
s_1r = s1_r.copy()
```

[19]:
```
s_2r = []
s_2r = s2_r.copy()
```

[20]:
```
s_3r = []
s_3r = s3_r.copy()
```

[21]:
```
sr = []
for i in s_1r:
    sr.append(i)
for i in s_2r:
    sr.append(i)
for i in s_3r:
    sr.append(i)
```

[22]:
```
for i in sr:
    a_set.add(i)
```

[23]:
```
a_set
```

[23]:
```
{AUX -> 'will',
 Det -> 'a',
 Det -> 'the',
 N -> 'Homer',
 N -> 'Marge',
 N -> 'donut',
 N -> 'ham',
 N -> 'sandwich',
 N -> 'table',
 NP -> Det N,
 NP -> Det N N,
 NP -> N,
 NP -> NP PP,
 P -> 'on',
 PP -> P NP,
 S -> AUX NP VP,
 S -> NP AUX VP,
 S -> NP VP,
 V -> 'ate',
 V -> 'make',
 VP -> V NP}
```

[24]:
```
more_r = []
more_r = list(a_set)
```

[25]:
```
more_r
```

[25]:
```
[Det -> 'a',
 S -> AUX NP VP,
 NP -> NP PP,
 N -> 'table',
 N -> 'sandwich',
 NP -> Det N,
 N -> 'Homer',
 P -> 'on',
 N -> 'ham',
 Det -> 'the',
 V -> 'make',
 AUX -> 'will',
 PP -> P NP,
 VP -> V NP,
 S -> NP AUX VP,
 N -> 'Marge',
 V -> 'ate',
 N -> 'donut',
 NP -> N,
 NP -> Det N N,
 S -> NP VP]
```

c. Add the additional rules to your grammar1's production rules, using the .extend() method.

[26]:
```
Grammar_1.productions().extend(list(more_r))
```

Marge will make a ham sandwich

[28]:
```
results = word_tokenize("")
par = nltk.ChartParser(Grammar_1)
for i in par.parse(results):
    print(i)
```

d. And then, you have to re-initialize the grammar using the extended production rules (highlighted part). An illustration:

[29]:
```
grammar3 = nltk.CFG.fromstring("""
S -> NP VP
NP -> N
VP -> V
N -> 'Homer'
V -> 'sleeps'
""")
```

[30]:
```
print(grammar3)
```

```
Grammar with 5 productions (start state = S)
    S -> NP VP
    NP -> N
    VP -> V
    N -> 'Homer'
    V -> 'sleeps'
```

[31]:
```
more_r
```

[31]:
```
[Det -> 'a',
 S -> AUX NP VP,
 NP -> NP PP,
 N -> 'table',
 N -> 'sandwich',
 NP -> Det N,
 N -> 'Homer',
 P -> 'on',
 N -> 'ham',
 Det -> 'the',
 V -> 'make',
 AUX -> 'will',
 PP -> P NP,
 VP -> V NP,
 S -> NP AUX VP,
 N -> 'Marge',
 V -> 'ate',
 N -> 'donut',
 NP -> N,
 NP -> Det N N,
 S -> NP VP]
```

e. Now, rebuild your chart parser with the updated grammar1. And try parsing the three sentences. It should successfully parse them.

[33]:
```
grammar3.productions().extend(more_r)
grammer3 = nltk.grammar.CFG(grammar3.start(), grammar3.productions())
print(grammer3)
```

```
Grammar with 26 productions (start state = S)
    S -> NP VP
    NP -> N
    VP -> V
    N -> 'Homer'
    V -> 'sleeps'
    Det -> 'a'
    S -> AUX NP VP
    NP -> NP PP
    N -> 'table'
    N -> 'sandwich'
```

```
NP -> Det N
N -> 'Homer'
P -> 'on'
N -> 'ham'
Det -> 'the'
V -> 'make'
AUX -> 'will'
PP -> P NP
VP -> V NP
S -> NP AUX VP
N -> 'Marge'
V -> 'ate'
N -> 'donut'
NP -> N
NP -> Det N N
S -> NP VP
```

0.1.4 5. Try parsing another sentence of your own that is covered by the newly extended grammar1. Are you satisfied with the result?. Also, compare the result with other parsers – Recursive Descent Parser and Shift Reduce Parser.

[34]:
```
Grammar_1 = nltk.CFG.fromstring("""
S -> NP VP | NP VP
NP -> N | Det N | PRO | N N
VP -> V NP CP | VP ADVP | V NP
ADVP -> ADV ADV
CP -> COMP S
N -> 'Lisa' | 'brother' | 'peanut' | 'butter'
V -> 'told' | 'liked'
COMP -> 'that'
Det -> 'her'
PRO -> 'she'
ADV -> 'very' | 'much'

S -> NP VP
NP -> NP CONJ NP | N | NP PP | Det N | N | Det N
VP -> VP PP | VP CONJ VP | V | V
PP -> P NP | P NP
N -> 'Homer' | 'friends' | 'work' | 'bar'
V -> 'drank' | 'sang'
CONJ -> 'and' | 'and'
Det -> 'his' | 'the'
P -> 'from' | 'in'

S -> NP VP
NP -> NP CONJ NP | N | N
VP -> V ADJP
```

```
ADJP -> ADJP CONJ ADJP | ADJ | ADV ADJ
N -> 'Homer' | 'Marge'
V -> 'are'
CONJ -> 'and' | 'but'
ADJ -> 'poor' | 'happy'
ADV -> 'very'

S -> NP VP | NP AUX VP
NP -> PRO | NP CP | Det N | PRO | PRO | PRO | N |Det N
VP -> V NP PP | V NP NP
CP -> COMP S
PP -> P NP
Det -> 'the' | 'his'
PRO -> 'he' | 'I' | 'him'
N -> 'book' | 't' | 'sister'
V -> 'gave' | 'given'
COMP -> 'that'
AUX -> 'had'
P -> 'to'

S -> NP VP
NP -> PRO | Det N | Det N
VP -> V NP PP
PP -> P NP
Det -> 'the' | 'his'
PRO -> 'he'
N -> 'book' | 'sister'
V -> 'gave'
P -> 'to'

S -> NP VP
NP -> Det ADJ N | Det ADJ ADJ N | N
VP -> V NP|VP PP
PP -> P NP
Det -> 'the' | 'the'
ADJ -> 'big' | 'tiny' | 'nerdy'
N -> 'bully' | 'kid' | 'school'
V -> 'punched'
P -> 'after'

S -> NP AUX VP
NP -> N |  Det N N
VP -> V NP
N -> 'Marge' | 'ham' | 'sandwich'
AUX -> 'will'
V -> 'make'
Det -> 'a'
```

```


S -> AUX NP VP
NP -> N | Det N N
VP -> V NP
N -> 'Marge'
V -> 'make'
AUX -> 'will'
Det -> 'a'
N -> 'Marge' | 'ham' | 'sandwich'


S -> NP VP
NP -> N | NP PP | Det N  | Det N
PP -> P NP
VP -> V NP
N -> 'Homer' | 'donut' | 'table'
V -> 'ate'
Det -> 'the' | 'the'
P -> 'on'
""")
```

[35]:
```
sent = word_tokenize("will Marge make a ham sandwich")
par = nltk.ChartParser(Grammar_1)
for i in par.parse(sent):
    print(i)
```

```
(S
  (AUX will)
  (NP (N Marge))
  (VP (V make) (NP (Det a) (N ham) (N sandwich))))
(S
  (AUX will)
  (NP (N Marge))
  (VP (V make) (NP (Det a) (N ham)) (NP (N sandwich))))
```

0.1.5 5. Try parsing another sentence of your own that is covered by the newly extended grammar1. Are you satisfied with the result?. Also, compare the result with other parsers – Recursive Descent Parser and Shift Reduce Parser.

[36]:
```
#sen = word_tokenize("will Marge make a ham sandwich")
#rd_par = nltk.RecursiveDescentParser(Grammar_1)
#for tree in rd_par.parse(sen):
    #print(tree)
```

[37]:
```
#sent = word_tokenize("will Marge make a ham sandwich")
#sr_par = nltk.ShiftReduceParser(Grammar_1)
```

```
#for tree in rd_par.parse(sent):
    #print(tree)
```

0.1.6 6. As the final step, pickle your grammar1 as lab12_grammar.pkl.

[38]:
```
import pickle
with open('lab12_grammar.pkl', 'wb') as f:
    pickle.dump(Grammar_1, f)
```

lab14_nlp_viviyan_33

June 8, 2021

0.0.1 Viviyan Richards W

205229133

0.1 Lab14. Word Sense Disambiguation with Improved Lesk Algorithm

0.1.1 EXERCISE-1

[1]:
```
import nltk
from nltk.wsd import lesk
from nltk.corpus import wordnet as wn
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data]   Unzipping corpora/wordnet.zip.
```

[1]:
```
True
```

[2]:
```
for ss in wn.synsets('bass'):
    print(ss,ss.definition())
```

```
Synset('bass.n.01') the lowest part of the musical range
Synset('bass.n.02') the lowest part in polyphonic music
Synset('bass.n.03') an adult male singer with the lowest voice
Synset('sea_bass.n.01') the lean flesh of a saltwater fish of the family
Serranidae
Synset('freshwater_bass.n.01') any of various North American freshwater fish
with lean flesh (especially of the genus Micropterus)
Synset('bass.n.06') the lowest adult male singing voice
Synset('bass.n.07') the member with the lowest range of a family of musical
instruments
Synset('bass.n.08') nontechnical name for any of numerous edible marine and
freshwater spiny-finned fishes
Synset('bass.s.01') having or denoting a low vocal or instrumental range
```

[3]:
```
print(lesk('I went fishing for some sea bass'.split(),'bass','n'))
```

```
Synset('bass.n.08')
```

[4]:
```
print(lesk('Avishai Cohen is an Israeli jazz musician. He plays double bass and␣
 ↪is also a composer'.split(), 'bass','n'))
```

```
Synset('sea_bass.n.01')
```

0.1.2 EXERCISE-2: Print senses for 'chair'

According to WordNet, how many distinct senses does 'chair' have? What are the hyponyms of 'chair' in its 'chair.n.01' sense? What is its hypernym, and what is its hyper-hypernym?

[5]:
```python
for ss in wn.synsets('chair'):
    print(ss,ss.definition())
```

```
Synset('chair.n.01') a seat for one person, with a support for the back
Synset('professorship.n.01') the position of professor
Synset('president.n.04') the officer who presides at the meetings of an
organization
Synset('electric_chair.n.01') an instrument of execution by electrocution;
resembles an ordinary seat for one person
Synset('chair.n.05') a particular seat in an orchestra
Synset('chair.v.01') act or preside as chair, as of an academic department in a
university
Synset('moderate.v.01') preside over
```

[6]:
```python
syn = wn.synsets('chair')[0]
print(syn)
```

```
Synset('chair.n.01')
```

[7]:
```python
print ("Synset name :  ", syn.name())

print ("\nSynset abstract term :  ", syn.hypernyms())

print ("\nSynset specific term :  ",
       syn.hypernyms()[0].hyponyms())

syn.root_hypernyms()

print ("\nSynset root hypernerm :  ", syn.root_hypernyms)
```

```
Synset name :   chair.n.01

Synset abstract term :   [Synset('seat.n.03')]

Synset specific term :   [Synset('bench.n.01'), Synset('bench.n.07'),
Synset('box.n.08'), Synset('box_seat.n.01'), Synset('chair.n.01'),
Synset('ottoman.n.03'), Synset('sofa.n.01'), Synset('stool.n.01'),
Synset('toilet_seat.n.01')]

Synset root hypernerm :   <bound method Synset.root_hypernyms of
Synset('chair.n.01')>
```

0.1.3 EXERCISE-3: Disambiguate the correct senses given the context sentence

[8]:
```
from nltk.corpus import wordnet as wn
from nltk.stem import PorterStemmer
from itertools import chain
bank_sents = ['I went to the bank to deposit my money', 'The river bank was␣
↪full of dead fishes']
plant_sents = ['The workers at the industrial plant were overworked','The plant␣
↪was no longer bearing flowers']
ps = PorterStemmer()
```

[9]:
```
def my_lesk(context_sentence, ambiguous_word,pos=None, stem=True,␣
↪hyperhypo=True):
    max_overlaps = 0
    lesk_sense = None
    context_sentence = context_sentence.split()
    for ss in wn.synsets(ambiguous_word):
    # If POS is specified.
    if pos and ss.pos is not pos:
        continue
    lesk_dictionary = []
    # Includes definition.
    defns = ss.definition().split()
    lesk_dictionary += defns
  # Includes lemma_names.
    lesk_dictionary += ss.lemma_names()
  # Optional: includes lemma_names of hypernyms and hyponyms.
    if hyperhypo == True:
        hhwords = ss.hypernyms() + ss.hyponyms()
    lesk_dictionary += list(chain(*[w.lemma_names() for w in hhwords] ))
  # Matching exact words causes sparsity, so lets match stems.
    if stem == True:
        lesk_dictionary = [ps.stem(w) for w in lesk_dictionary]
    context_sentence = [ps.stem(w) for w in context_sentence]
    overlaps = set(lesk_dictionary).intersection(context_sentence)
    if len(overlaps) > max_overlaps:
        lesk_sense = ss
    max_overlaps = len(overlaps)
    return lesk_sense
```

[10]:
```
# evaluate senses
print("Context:", bank_sents[0])
answer = my_lesk(bank_sents[0],'bank')
print("Sense:", answer)
print("Definition:",answer.definition)
```

```
Context: I went to the bank to deposit my money
Sense: Synset('bank.v.07')
```

```
Definition: <bound method Synset.definition of Synset('bank.v.07')>
```

[11]:
```
print("Context:", bank_sents[1])
answer = my_lesk(bank_sents[1],'bank')
print("Sense:", answer)
print("Definition:", answer.definition)
```

```
Context: The river bank was full of dead fishes
Sense: Synset('bank.v.07')
Definition: <bound method Synset.definition of Synset('bank.v.07')>
```

[12]:
```
print("Context:", plant_sents[0])
answer = my_lesk(plant_sents[0],'plant')
print("Sense:", answer)
print("Definition:",answer.definition)
```

```
Context: The workers at the industrial plant were overworked
Sense: Synset('plant.v.06')
Definition: <bound method Synset.definition of Synset('plant.v.06')>
```

0.1.4 EXERCISE-4

Learn further examples for synsets at https://www.programcreek.com/python/example/91604/nltk.corpus.wordne

lab15_nlp_viviyan_33

June 8, 2021

0.0.1 viviyan richards W

205229133

0.1 Lab15. Text Processing using SpaCy

0.1.1 EXERCISES

0.1.2 Question 1. Print the tokens of the string, "welcome all of you for this NLP with spacy course"

[1]:
```
import spacy
nlp = spacy.load("en_core_web_sm")
```

[2]:
```
doc = nlp("welcome all of you for this NLP with spacy course")
for token in doc:
    print(token.text, token.pos_, token.dep_)
```

```
welcome VERB ROOT
all DET dobj
of ADP prep
you PRON pobj
for ADP prep
this DET det
NLP PROPN pobj
with ADP prep
spacy NOUN compound
course NOUN pobj
```

0.1.3 Question 2. Create a text file that contains the above string, open that file and print the tokens

[2]:
```

```

0.1.4 Question 3. Consider the following sentences and print each sentence in one line

[3]:
```python
my_text = ('Rajkumar Kannan is a ML developer currently'
    ' working for a London-based Edtech'
    ' company. He is interested in learning'
    ' Natural Language Processing.'
    ' He keeps organizing local Python meetups'
    ' and several internal talks at his workplace.')
```

0.1.5 Question 4. For the list of strings from my_text, print the following for each token:

[4]:
```python
doc=nlp(my_text)
for token in doc:
    print(token.text, token.lemma_, token.pos_, token.tag_, token.dep_,
            token.shape_, token.is_alpha, token.is_stop)
```

```
Rajkumar Rajkumar PROPN NNP compound Xxxxx True False
Kannan Kannan PROPN NNP nsubj Xxxxx True False
is be AUX VBZ ROOT xx True True
a a DET DT det x True True
ML ML PROPN NNP compound XX True False
developer developer NOUN NN attr xxxx True False
currently currently ADV RB advmod xxxx True False
working work VERB VBG acl xxxx True False
for for ADP IN prep xxx True True
a a DET DT det x True True
London London PROPN NNP npadvmod Xxxxx True False
- - PUNCT HYPH punct - False False
based base VERB VBN amod xxxx True False
Edtech Edtech PROPN NNP compound Xxxxx True False
company company NOUN NN pobj xxxx True False
. . PUNCT . punct . False False
He -PRON- PRON PRP nsubj Xx True True
is be AUX VBZ ROOT xx True True
interested interested ADJ JJ acomp xxxx True False
in in ADP IN prep xx True True
learning learn VERB VBG pcomp xxxx True False
Natural Natural PROPN NNP compound Xxxxx True False
Language Language PROPN NNP compound Xxxxx True False
Processing Processing PROPN NNP dobj Xxxxx True False
. . PUNCT . punct . False False
He -PRON- PRON PRP nsubj Xx True True
keeps keep VERB VBZ ROOT xxxx True False
organizing organize VERB VBG xcomp xxxx True False
local local ADJ JJ amod xxxx True False
Python Python PROPN NNP compound Xxxxx True False
```

```
meetups meetup NOUN NNS dobj xxxx True False
and and CCONJ CC cc xxx True True
several several ADJ JJ amod xxxx True True
internal internal ADJ JJ amod xxxx True False
talks talk NOUN NNS conj xxxx True False
at at ADP IN prep xx True True
his -PRON- DET PRP$ poss xxx True True
workplace workplace NOUN NN pobj xxxx True False
. . PUNCT . punct . False False
```

0.1.6 Question 5. Detect and print hyphenated words from my_text. For example, London-based.

[5]:
```python
import re
import spacy
from spacy.tokenizer import Tokenizer
from spacy.util import compile_prefix_regex, compile_infix_regex,␣
↪compile_suffix_regex

def custom_tokenizer(nlp):
    infix_re = re.compile(r'''[.\,\?\:\;\...\‘\’\`\"\"\"\'~]''')
    prefix_re = compile_prefix_regex(nlp.Defaults.prefixes)
    suffix_re = compile_suffix_regex(nlp.Defaults.suffixes)

    return Tokenizer(nlp.vocab, prefix_search=prefix_re.search,
                                suffix_search=suffix_re.search,
                                infix_finditer=infix_re.finditer,
                                token_match=None)

nlp = spacy.load('en')
nlp.tokenizer = custom_tokenizer(nlp)
```

[6]:
```python
doc = nlp(my_text)
[token.text for token in doc]
```

[6]:
```
['Rajkumar',
 'Kannan',
 'is',
 'a',
 'ML',
 'developer',
 'currently',
 'working',
 'for',
 'a',
 'London-based',
 'Edtech',
```

```
 'company',
 '.',
 'He',
 'is',
 'interested',
 'in',
 'learning',
 'Natural',
 'Language',
 'Processing',
 '.',
 'He',
 'keeps',
 'organizing',
 'local',
 'Python',
 'meetups',
 'and',
 'several',
 'internal',
 'talks',
 'at',
 'his',
 'workplace',
 '.']
```

0.1.7 Question 6. Print all stop words defined in SpaCy

[7]:
```python
print(nlp.Defaults.stop_words)
```

```
{'do', '‘m', 'towards', 'to', 'really', 'too', 'take', 'below', 'no', 'along',
'as', 'together', 'noone', 'five', 'whence', 'about', 'nobody', 'top', 'none',
'since', 'thereupon', 'nor', 'first', 'off', 'not', 'never', 'hence', 'last',
'me', 'be', 'seems', 'put', 'does', 'over', 'three', 'ten', 'ca', 'always',
'whither', 'eleven', '’s', 'across', 'least', 'bottom', 'at', 'whereafter',
'fifteen', 'whereas', 'been', 'herein', 'sometime', 'either', 'ever', '’m',
'sixty', 'could', 'whoever', 'against', 'thereby', 'has', '‘ll', '‘d', 'any',
'becomes', '’ve', 'hers', 'myself', 'still', 'seem', 'because', 'upon', 'into',
'even', 'him', 'whereupon', 'each', 'but', 'therefore', 'n‘t', 'doing', 'why',
'behind', 'became', 'would', 'several', 'twenty', 'our', 'due', 'might', 'was',
'toward', 'which', 'regarding', 'move', 'keep', 'us', 'being', 'n’t', 'its',
'else', 'while', 'your', 'wherein', 'yet', 'when', 'should', 'per', 'meanwhile',
'is', 'will', 'ours', 'around', 'mine', 'a', 'elsewhere', 'ourselves', 'thus',
'if', 'his', "'s", 'full', 'nothing', 'above', 'say', 'six', 'had', 'wherever',
'throughout', 'we', 'whole', 'whom', 'anything', 'who', 'them', 'beforehand',
'just', 'another', 're', 'for', 'name', 'everywhere', 'whether', 'herself',
'almost', "'ll", 'without', 'two', '’d', 'an', 'made', 'themselves', 'hereby',
'using', '’re', 'further', 'therein', 'you', 'alone', 'much', 'seeming', 'onto',
```

```
'did', 'well', 'where', 'twelve', 'indeed', "'ve", 'that', 'whereby', 'it',
'himself', 'i', 'side', 'somehow', 'so', 'by', 'amongst', 'until', 'then',
'nevertheless', 'those', 'and', "n't", 'own', 'same', 'show', 'many', 'yours',
'few', "'re", 'were', 'give', 'within', 'serious', 'under', 'also', 'thru',
'whose', 'again', '’ll', 'amount', 'fifty', 'before', 'may', 'neither',
'enough', 'otherwise', 'on', 'yourselves', 'anyhow', 'becoming', "'d",
'although', 'nowhere', 'down', 'mostly', 'her', 'she', 'can', 'former',
'hereafter', 'have', 'after', 'four', 'quite', 'latter', 'once', 'between',
'used', 'both', 'some', 'next', 'seemed', 'whenever', 'please', 'all', 'beyond',
'formerly', 'something', 'thereafter', 'my', 'part', 'everyone', 'back',
'thence', 'anywhere', 'already', 'other', 'their', 'during', 'go', 'the',
'what', 'of', 'now', 'less', 'beside', 'done', 'than', 'whatever', 'anyway',
'sometimes', 'get', 'often', 'latterly', '‘s', 'afterwards', 'one', 'more',
'must', 'very', '‘re', 'unless', 'he', "'m", 'how', 'except', 'in', 'up',
'empty', 'yourself', 'with', 'besides', 'every', 'cannot', 'third', 'such',
'there', 'via', 'are', 'rather', 'see', 'among', 'moreover', 'namely', 'call',
'this', 'hundred', 'hereupon', 'through', 'others', 'though', 'eight',
'everything', 'they', 'however', 'various', 'become', 'here', 'somewhere',
'itself', 'someone', 'or', 'nine', 'make', 'only', 'perhaps', 'most', 'anyone',
'forty', 'out', 'from', 'these', 'am', '‘ve', 'front'}
```

0.1.8 Question 7. Remove all stop words and print the rest of tokens from, my_text

[8]:
```
all_stopwords = nlp.Defaults.stop_words
[token.text for token in doc if not token.text in all_stopwords]
```

[8]:
```
['Rajkumar',
 'Kannan',
 'ML',
 'developer',
 'currently',
 'working',
 'London-based',
 'Edtech',
 'company',
 '.',
 'He',
 'interested',
 'learning',
 'Natural',
 'Language',
 'Processing',
 '.',
 'He',
 'keeps',
 'organizing',
 'local',
```

```
 'Python',
 'meetups',
 'internal',
 'talks',
 'workplace',
 '.']
```

0.1.9 Question 8. Print all lemma from my_text

[9]:
```
for token in doc:
    print(token, token.lemma_)
```

```
Rajkumar Rajkumar
Kannan Kannan
is be
a a
ML ML
developer developer
currently currently
working work
for for
a a
London-based London-based
Edtech Edtech
company company
. .
He -PRON-
is be
interested interested
in in
learning learn
Natural Natural
Language Language
Processing Processing
. .
He -PRON-
keeps keep
organizing organize
local local
Python Python
meetups meetup
and and
several several
internal internal
talks talk
at at
his -PRON-
workplace workplace
```

. .

0.1.10 Question 9. Perform Part of Speech Tagging on my_text and print the following tag informations token, token.tag_, token.pos_, spacy.explain(token.tag_)

[10]:
```
doc=nlp(my_text)
for token in doc:
    print(token.text, token.pos_, token.tag,spacy.explain(token.tag_))
```

```
Rajkumar PROPN 15794550382381185553 noun, proper singular
Kannan PROPN 15794550382381185553 noun, proper singular
is AUX 13927759927860985106 verb, 3rd person singular present
a DET 15267657372422890137 determiner
ML PROPN 15794550382381185553 noun, proper singular
developer NOUN 15308085513773655218 noun, singular or mass
currently ADV 164681854541413346 adverb
working VERB 1534113631682161808 verb, gerund or present participle
for ADP 1292078113972184607 conjunction, subordinating or preposition
a DET 15267657372422890137 determiner
London-based PROPN 15794550382381185553 noun, proper singular
Edtech PROPN 15794550382381185553 noun, proper singular
company NOUN 15308085513773655218 noun, singular or mass
. PUNCT 12646065887601541794 punctuation mark, sentence closer
He PRON 13656873538139661788 pronoun, personal
is AUX 13927759927860985106 verb, 3rd person singular present
interested ADJ 10554686591937588953 adjective
in ADP 1292078113972184607 conjunction, subordinating or preposition
learning VERB 1534113631682161808 verb, gerund or present participle
Natural PROPN 15794550382381185553 noun, proper singular
Language PROPN 15794550382381185553 noun, proper singular
Processing PROPN 15794550382381185553 noun, proper singular
. PUNCT 12646065887601541794 punctuation mark, sentence closer
He PRON 13656873538139661788 pronoun, personal
keeps VERB 13927759927860985106 verb, 3rd person singular present
organizing VERB 1534113631682161808 verb, gerund or present participle
local ADJ 10554686591937588953 adjective
Python PROPN 15794550382381185553 noun, proper singular
meetups NOUN 783433942507015291 noun, plural
and CCONJ 17571114184892886314 conjunction, coordinating
several ADJ 10554686591937588953 adjective
internal ADJ 10554686591937588953 adjective
talks NOUN 783433942507015291 noun, plural
at ADP 1292078113972184607 conjunction, subordinating or preposition
his DET 4062917326063685704 pronoun, possessive
workplace NOUN 15308085513773655218 noun, singular or mass
. PUNCT 12646065887601541794 punctuation mark, sentence closer
```

0.1.11 Question 10. How many NOUN and ADJ are there in my_text?. Print them and its count.

[11]:
```
nouns = []
for token in doc:
    if token.pos_ == 'NOUN':
        nouns.append(token)
print(len(nouns),nouns)
```

```
5 [developer, company, meetups, talks, workplace]
```

[12]:
```
adjectives = []
for token in doc:
    if token.pos_ == 'ADJ':
        adjectives.append(token)
print(len(adjectives),adjectives)
```

```
4 [interested, local, several, internal]
```

0.1.12 Question 11. Visualize POS tags of a sentence, my_text, using displaCy

[13]:
```
from spacy import displacy
displacy.render(doc, style='dep',jupyter=True)
```

```
<IPython.core.display.HTML object>
```

0.1.13 Question 12. Extract and print First Name and Last Name from my_text using Matcher.

[14]:
```
from spacy.matcher import Matcher
from spacy.tokens import Span
matcher = Matcher(nlp.vocab)
matcher.add("PERSON", [[{"lower": "rajkumar"}, {"lower": "kannan"}]])
matches = matcher(doc)
for match_id, start, end in matches:
    # Create the matched span and assign the match_id as a label
    span = Span(doc, start, end, label=match_id)
    print(span.text, span.label_)
```

```
Rajkumar Kannan PERSON
```

0.1.14 Question 13. Print the dependency parse tag values for the text, "Rajkumar is learning piano". Also, display dependency parse tree using displaCy.

[15]:
```
doc = nlp(u'Rajkumar is learning piano')
for token in doc:
    print(token.text, token.dep_)
displacy.render(doc, style='dep',jupyter=True)
```

```
Rajkumar nsubj
is aux
learning ROOT
piano dobj
```

```
<IPython.core.display.HTML object>
```

0.1.15 Question 14. Consider the following string.

a. Print the children of `developer`

[16]:
```python
d_text = 'Sam Peter is a Python developer currently working for a London-based␣
↪Fintech company'
doc = nlp(d_text)
[t.text for t in doc[5].children]
```

[16]:
```
['a', 'Python', 'working']
```

b. Print the previous neighboring node of `developer`

[17]:
```python
print (doc[5].nbor(-1))
```

```
Python
```

c. Print the next neighboring node of `developer`

[18]:
```python
print (doc[5].nbor())
```

```
currently
```

d. Print the all tokens on the left of `developer`

[19]:
```python
[t.text for t in doc[5].lefts]
```

[19]:
```
['a', 'Python']
```

e. Print the tokens on the right of `developer`

[20]:
```python
[t.text for t in doc[5].rights]
```

[20]:
```
['working']
```

f. Print the Print subtree of `developer`

[21]:
```python
[t.text for t in doc[5].subtree]
```

[21]:
```
['a',
 'Python',
 'developer',
 'currently',
 'working',
 'for',
 'a',
 'London-based',
```

'Fintech',
 'company']

0.1.16 Question 15. Print all Noun Phrases in the text

[22]:
```
conference_text = ('There is a developer conference happening on 21 July 2020␣
↪in New Delhi.')
conference_doc = nlp(conference_text)
for chunk in conference_doc.noun_chunks:
    print (chunk)
```

```
a developer conference
21 July
New Delhi
```

0.1.17 Question 16. Print all Verb Phrases in the text (you need to install textacy)

[22]:
```
#import spacy,en_core_web_sm
#import textacy
#about_talk_text = ('The talk will introduce reader about Use'
                   #' cases of Natural Language Processing in'
                   #' Fintech')
#pattern = r'(<VERB>?<ADV>*<VERB>+)'
#about_talk_doc = textacy.make_spacy_doc(about_talk_text, lang='en_core_web_sm')
#verb_phrases = textacy.extract.pos_regex_matches(about_talk_doc, pattern)

#for chunk in verb_phrases:
    #print(chunk.text)
#for chunk in about_talk_doc.noun_chunks:
    #print (chunk)
```

0.1.18 Question 17. Print all Named Entities in the text

[23]:
```
piano_class_text = ('Great Piano Academy is situated'
    ' in Mayfair or the City of London and has'
    ' world-class piano instructors.')
piano_class_doc = nlp(piano_class_text)
for ent in piano_class_doc.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_, spacy.explain(ent.
↪label_))
```

```
Great Piano Academy 0 19 ORG Companies, agencies, institutions, etc.
Mayfair 35 42 GPE Countries, cities, states
the City of London 46 64 GPE Countries, cities, states
```