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 [15]:

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

STEP 2: build goog2w_list

In [16]:

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

In [17]:

```
nimi = text[:]
```

In [18]:

```
mini[0].split()
```

Out[18]:

```
['0Uplink', 'verified', '523545']
```

In [19]:

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

Out[19]:

```
[(('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 [20]:

```
mini_list[0]
```

Out[20]:

```
(('0Uplink', 'verified'), 523545)
```

In [21]:

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

Out[21]:

```
[(('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),
```

In [22]:

```
goog2w_list[0]
```

Out[22]:

```
(('0Uplink', 'verified'), 523545)
```

STEP 3: build goog2w_fd

In [ ]:

```
!pip install nltk
```

In [10]:

```
import nltk
nltk.download('wordnet')
nltk.download('punkt')
from nltk import *
```

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

In [23]:

```
goog2w_fd = nltk.FreqDist()
goog2w_fd
```

Out[23]:

```
FreqDist({})
```

In [24]:

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

In [25]:

```
goog2w_fd[('of', 'the')]
```

Out[25]:

```
'2766332391'
```

In [26]:

```
goog2w_fd[('so', 'beautiful')]
```

Out[26]:

```
'612472'
```

STEP 4: explore

1. What are the top-10 bigrams?

In [27]:

```
goog2w_fd.most_common(10)
```

Out[27]:

```
[(('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 [28]:

```
import pickle as pkl
```

In [146]:

```
with open('goog2w_list.pkl', 'ab') as handle:
    pkl.dump(goog2w_list,handle)
```

In [147]:

```
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 [31]:

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

In [32]:

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

In [33]:

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

B. builds a list of individual sentences,

In [34]:

```
from nltk.tokenize import sent_tokenize as st
```

In [35]:

```
st(cona)
```

...

In [36]:

```
st(conp)
```

...

In [37]:

```
st(cons)
```

...

C. prints out how many sentences there are,

In [38]:

```
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 [39]:

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

In [40]:

```
t1=word_tokenize(cona)
print(t1)
```

...

In [41]:

```
t2=word_tokenize(conp)
print(t2)
```

. . .

In [42]:

```
t3 = word_tokenize(cons)
print(t3)
```

. . .

F. builds a frequency count dictionary of words,

In [43]:

```
from nltk import *
```

In [44]:

```
da1 = FreqDist(t1)
da1
```

Out[44]:

```
FreqDist({',': 12016, '.': 6355, 'to': 5125, 'the': 4844, 'and': 4653, 'of':
4272, 'I': 3177, '--': 3100, 'a': 3001, "'": 2452, ...})
```

In [45]:

```
da2 = FreqDist(t2)
da2
```

Out[45]:

```
FreqDist({',': 7024, 'the': 3119, '.': 3119, 'to': 2751, 'and': 2724, 'of':
2562, 'a': 1528, 'in': 1340, 'was': 1330, ';': 1319, ...})
```

In [46]:

```
da3 = FreqDist(t3)
da3
```

Out[46]:

```
FreqDist({',': 9901, 'to': 4050, '.': 4023, 'the': 3860, 'of': 3564, 'and':
3348, 'her': 2434, 'a': 2025, 'I': 2003, 'in': 1873, ...})
```

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

In [47]:

```
da1.most_common(50)
```

. . .

In [48]:

```
da2.most_common(50)
```

...

In [49]:

```
da3.most_common(50)
```

...

EXCERCISE 3

A. imports necessary modules,

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

In [50]:

```
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 [51]:

```
tokenizer = nltk.tokenize.WhitespaceTokenizer()
tok = tokenizer.tokenize(nov)
tok
```

Out[51]:

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

In [53]:

```
b2 = list(nltk.bigrams(tok))
b2fd = nltk.FreqDist(b2)
b2fd
```

Out[53]:

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

In [55]:

```
import re
from collections import Counter
```

In [56]:

```
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',): 29, ('so soon',): 23, ('so good',): 20, ('so often',): 16, ('so kin
d',): 14, ('so great',): 14, ('so it',): 14, ('so entirely',): 11, ('so ha
ppy',): 11, ('so you',): 11, ('so near',): 11, ('so to',): 10, ('so anxiou
s',): 10, ('so easily',): 9, ('so she',): 9, ('so glad',): 9, ('so fon
d',): 8, ('so ill',): 8, ('so strong',): 8, ('so bad',): 7, ('so as',): 7,
('so lately',): 7, ('so miserable',): 7, ('so young',): 7, ('so totall
y',): 6, ('so truly',): 6, ('so short',): 6, ('so few',): 6, ('so that',):
6, ('so particularly',): 6, ('so full',): 6, ('so large',): 6, ('so extrem
ely',): 6, ('so cheerful',): 6, ('so pleasantly',): 5, ('so interestin
g',): 5, ('so completely',): 5, ('so fast',): 5, ('so obliging',): 5, ('so
lovely',): 5, ('so at',): 5, ('so suddenly',): 5, ('so agreeable',): 5,
('so dear',): 4, ('so proper',): 4, ('so busy',): 4, ('so forth',): 4, ('s
o warmly',): 4, ('so charming',): 4, ('so with',): 4, ('so deceived',): 4,
('so odd',): 4, ('so pleased',): 4, ('so delighted',): 4, ('so happene
d',): 4, ('so thoroughly',): 4, ('so sudden',): 4, ('so on',): 4, ('so lib
eral',): 4, ('so attentive',): 4, ('so he',): 4, ('so sorry',): 4, ('so sh
ocked',): 4, ('so wretched',): 4, ('so highly',): 4, ('so determined',):
```

1. a_toks: word tokens, all in lowercase

In [57]:

```
tokenizer = nltk.tokenize.WhitespaceTokenizer()
a_toks = tokenizer.tokenize(nov.lower())
a_toks
```

Out[57]:

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

2. a_tokfd: word frequency distribution

In [58]:

```
a_tokfd = FreqDist(a_toks)
a_tokfd
```

Out[58]:

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

3. a_bigrams: word bigrams, cast as a list

In [59]:

```
a_bigrams = list(nltk.bigrams(a_toks))
a_bigrams
```

Out[59]:

```
[('[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'),
```

4. a_bigramfd: bigram frequency distribution

In [103]:

```
a_bigramfd = nltk.FreqDist(a_bigrams)
a_bigramfd
```

Out[103]:

```
FreqDist({('of', 'the'): 1411, ('to', 'be'): 1342, ('in', 'the'): 1115, ('i
t', 'was'): 826, ('she', 'had'): 715, ('had', 'been'): 669, ('to', 'the'): 6
50, ('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 [104]:

```
from nltk.probability import ConditionalFreqDist
from nltk.tokenize import word_tokenize
```

In [105]:

```
a_bigramcfd = ConditionalFreqDist()
```

In [106]:

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

In [107]:

```
a_bigramcfd
```

Out[107]:

```
<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 [145]:

```
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 [108]:

```
len(a_toks)
```

Out[108]:

```
360148
```

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

In [109]:

```
ws=a_tokfd.most_common(20)
n = dict(ws)
n
```

Out[109]:

```
{'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 [110]:

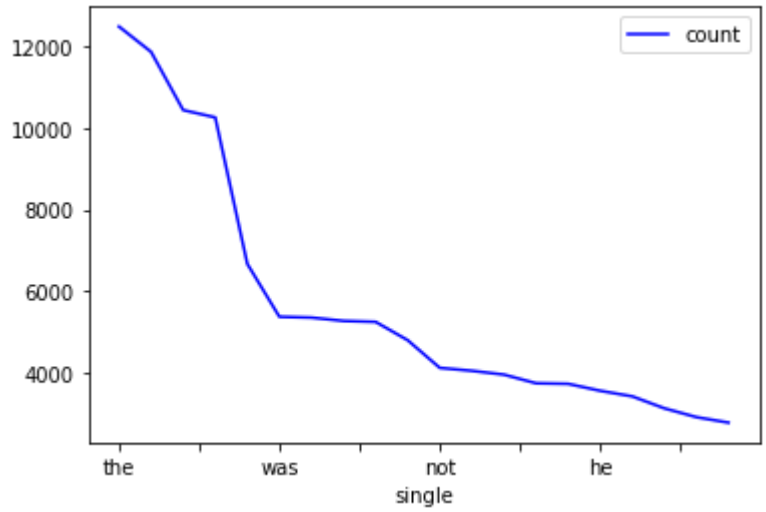
```
df = pd.DataFrame(list(n.items()))
df.columns = ['single','count']
df
```

Out[110]:

	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 [111]:

```
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 [112]:

```
v=a_bigramfd.most_common(20)
m = dict(v)
m
```

Out[112]:

```
{('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 [113]:

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

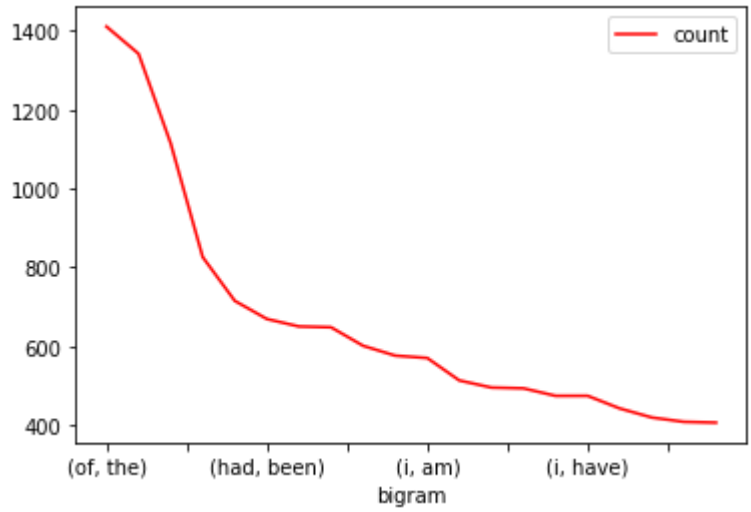
Out[113]:

	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 [115]:

```
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 [123]:

```
so_count=a_tokfd['so']
print(so_count)

tot=len(a_tokfd)
print(tot)

rel_freq = so_count/tot
rel_freq
```

Out[123]:

```
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 [127]:

```
ab.most_common(20)
```

Out[127]:

```
[(('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 [131]:

```
ab_dict = dict(ab)
ab_dict
```

...

In [138]:

```
tot_occ=len(ab_dict)
tot_occ
```

Out[138]:

```
584
```

In [141]:

```
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 [142]:

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

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