CIS 668 Assignment #2

Analysis of Review Contents

***Instructor**: Professor Lu Xiao*

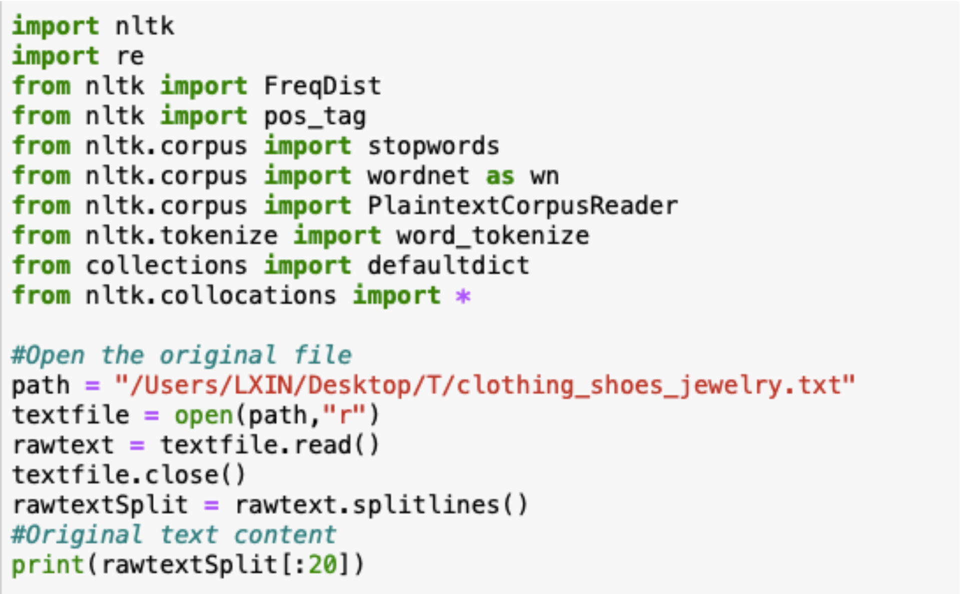
***Student**: Leah Luo (NetID lluojr)*

***Date**: 03/06/2020*

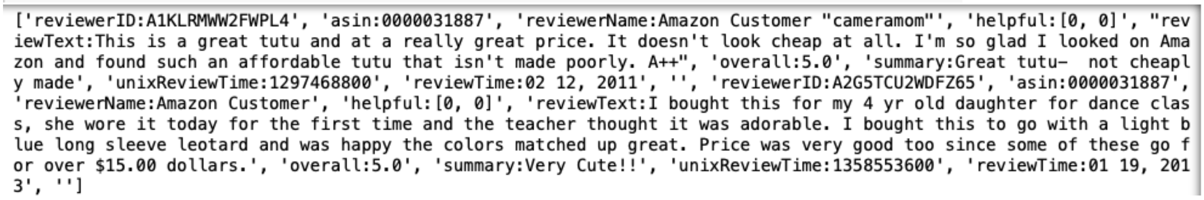
CIS 668
Leah Luo (SUID 326896495, NetID lluojr)
Assignment #2
03/06/2020

Cleaning Process

1. **Retrieve the data needed.**
   a. Extract the text content from the text file. (rawtextSplit)
   The second figure shows the first 20 words in the original file.

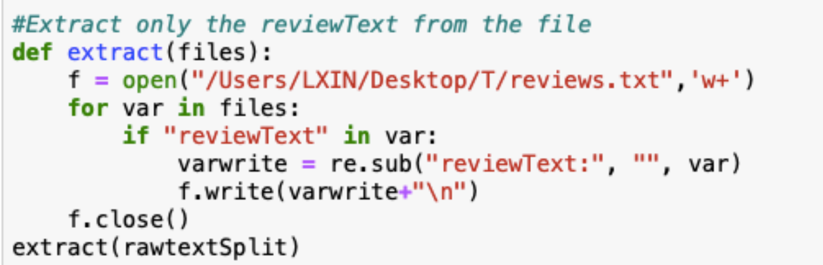


```
import nltk
import re
from nltk import FreqDist
from nltk import pos_tag
from nltk.corpus import stopwords
from nltk.corpus import wordnet as wn
from nltk.corpus import PlaintextCorpusReader
from nltk.tokenize import word_tokenize
from collections import defaultdict
from nltk.collocations import *

#Open the original file
path = "/Users/LXIN/Desktop/T/clothing_shoes_jewelry.txt"
textfile = open(path,"r")
rawtext = textfile.read()
textfile.close()
rawtextSplit = rawtext.splitlines()
#Original text content
print(rawtextSplit[:20])
```

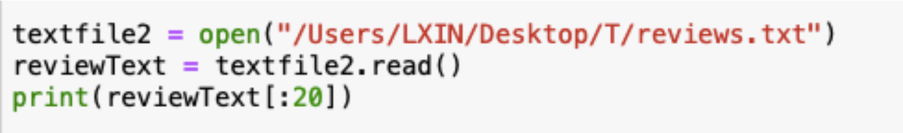


```
['reviewerID:A1KLRMWW2FWPL4', 'asin:0000031887', 'reviewerName:Amazon Customer "cameramom"', 'helpful:[0, 0]', "rev
iewText:This is a great tutu and at a really great price. It doesn't look cheap at all. I'm so glad I looked on Ama
zon and found such an affordable tutu that isn't made poorly. A++", 'overall:5.0', 'summary:Great tutu-  not cheapl
y made', 'unixReviewTime:1297468800', 'reviewTime:02 12, 2011', '', 'reviewerID:A2G5TCU2WDFZ65', 'asin:0000031887',
'reviewerName:Amazon Customer', 'helpful:[0, 0]', 'reviewText:I bought this for my 4 yr old daughter for dance clas
s, she wore it today for the first time and the teacher thought it was adorable. I bought this to go with a light b
lue long sleeve leotard and was happy the colors matched up great. Price was very good too since some of these go f
or over $15.00 dollars.', 'overall:5.0', 'summary:Very Cute!!', 'unixReviewTime:1358553600', 'reviewTime:01 19, 201
3', '']
```

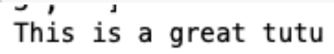
   b. Create a document to save the extracted text, which contains only the reviews contents (reviews.txt)
   The second figure shows the first 20 words in the new file.



```
#Extract only the reviewText from the file
def extract(files):
    f = open("/Users/LXIN/Desktop/T/reviews.txt",'w+')
    for var in files:
        if "reviewText" in var:
            varwrite = re.sub("reviewText:", "", var)
            f.write(varwrite+"\n")
    f.close()
extract(rawtextSplit)
```

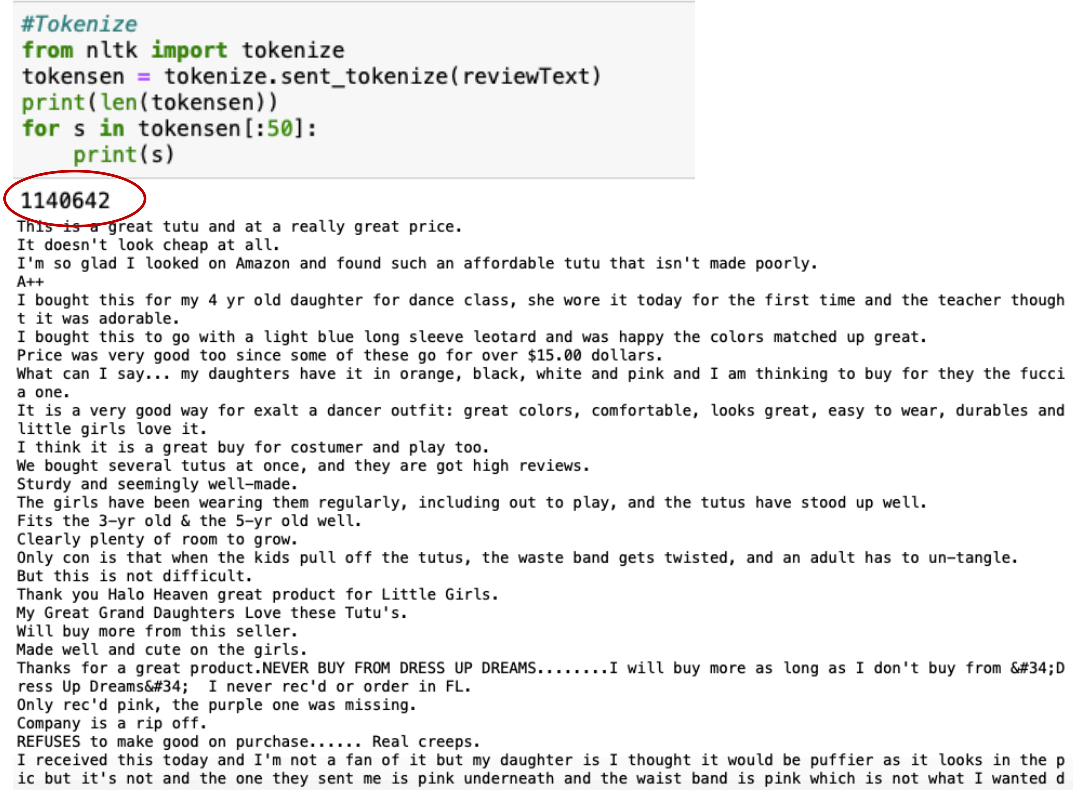


```
textfile2 = open("/Users/LXIN/Desktop/T/reviews.txt")
reviewText = textfile2.read()
print(reviewText[:20])
```



```
This is a great tutu
```

2. **Pre-processing the data**
    a. Tokenization. Open the new text file *reviews.* Separate the file content into tokens with sentence tokenizer.
    Output: number of sentences in the file (length) and the first few sentences.



```
#Tokenize
from nltk import tokenize
tokensen = tokenize.sent_tokenize(reviewText)
print(len(tokensen))
for s in tokensen[:50]:
    print(s)
```

```
1140642
This is a great tutu and at a really great price.
It doesn't look cheap at all.
I'm so glad I looked on Amazon and found such an affordable tutu that isn't made poorly.
A++
I bought this for my 4 yr old daughter for dance class, she wore it today for the first time and the teacher though
t it was adorable.
I bought this to go with a light blue long sleeve leotard and was happy the colors matched up great.
Price was very good too since some of these go for over $15.00 dollars.
What can I say... my daughters have it in orange, black, white and pink and I am thinking to buy for they the fucci
a one.
It is a very good way for exalt a dancer outfit: great colors, comfortable, looks great, easy to wear, durables and
little girls love it.
I think it is a great buy for costumer and play too.
We bought several tutus at once, and they are got high reviews.
Sturdy and seemingly well-made.
The girls have been wearing them regularly, including out to play, and the tutus have stood up well.
Fits the 3-yr old & the 5-yr old well.
Clearly plenty of room to grow.
Only con is that when the kids pull off the tutus, the waste band gets twisted, and an adult has to un-tangle.
But this is not difficult.
Thank you Halo Heaven great product for Little Girls.
My Great Grand Daughters Love these Tutu's.
Will buy more from this seller.
Made well and cute on the girls.
Thanks for a great product.NEVER BUY FROM DRESS UP DREAMS........I will buy more as long as I don't buy from &#34;D
ress Up Dreams&#34;  I never rec'd or order in FL.
Only rec'd pink, the purple one was missing.
Company is a rip off.
REFUSES to make good on purchase...... Real creeps.
I received this today and I'm not a fan of it but my daughter is I thought it would be puffier as it looks in the p
ic but it's not and the one they sent me is pink underneath and the waist band is pink which is not what I wanted d
```

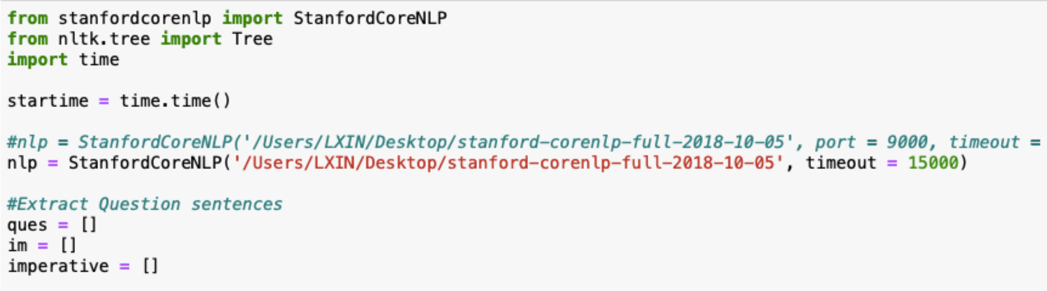
3. **Extract question sentences includes Adjective phrases**
    a. Download and install Stanford CoreNLP tool
    Reference: < [*https://stanfordnlp.github.io/CoreNLP/*](https://stanfordnlp.github.io/CoreNLP/)>
        i. What is Stanford CoreNLP: It is a Java suite of core NLP tools. It can mark up the structure of sentences in terms of phrases and syntactic dependencies.
        ii. Why Stanford CoreNLP: It can parse sentences and mark up the structure fast and efficient, which can save a lot time comparing with building our own CFG grammar rules and then analyze the sentences.
        iii. How does it run: Note that you can just simply import it in the python program and run. I run the program in kernel mode. Another way is run the Stanford CoreNLP in terminal, but some more parameters need to be set.
    b. Structure of question sentence
        i. We need to understand the grammar and form of an interrogative (question) sentence before applying certain rules. The link is the Englishclub website, which gives the definition and form of the question sentence. <[*https://www.englishclub.com/grammar/sentence/type-interrogative.htm*](https://www.englishclub.com/grammar/sentence/type-interrogative.htm)>
        ii. Parsing question sentences with Stanford parser to find the tag assigned to question sentences and two specific tags are found: “SQ” and “SBARQ”.

Note that “SQ” denotes “Inverted yes/no question, or main clause of a wh-question”, while “SBARQ” denotes “Direct question introduced by a wh-word or a wh-phrase”.

iii. According to the definition of question sentence, the final punctuation is always a question mark(?).

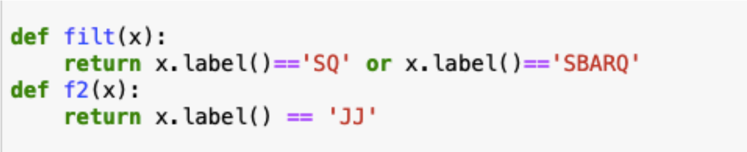
c. Extract the question sentences

i. Import StanfordCoreNLP and nltk.tree



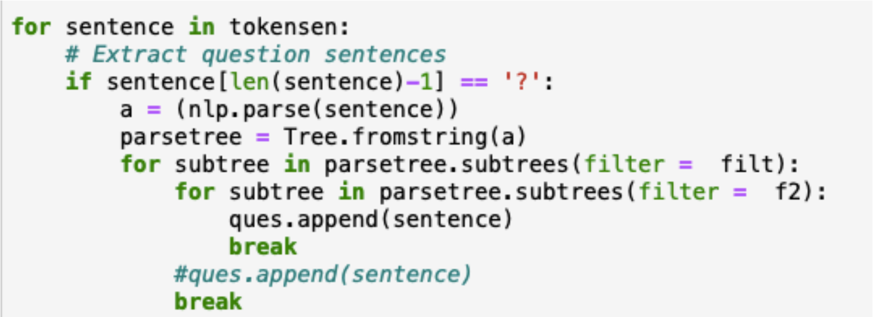
```
from stanfordcorenlp import StanfordCoreNLP
from nltk.tree import Tree
import time

startime = time.time()

#nlp = StanfordCoreNLP('/Users/LXIN/Desktop/stanford-corenlp-full-2018-10-05', port = 9000, timeout =
nlp = StanfordCoreNLP('/Users/LXIN/Desktop/stanford-corenlp-full-2018-10-05', timeout = 15000)

#Extract Question sentences
ques = []
im = []
imperative = []
```

ii. Define function to find sentences have “SQ” or “SBARQ” tag, and function to extract sentences with adjective phrase.



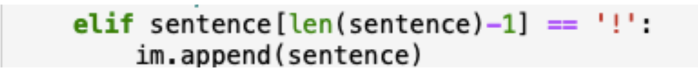
```
def filt(x):
    return x.label()=='SQ' or x.label()=='SBARQ'
def f2(x):
    return x.label() == 'JJ'
```

iii. Loop every sentence, use Stanford parse to get the sentence structure in terms of phrases if the sentence ends with question mark.
Use nltk.tree to parse the structure, which creates a tree structure for storing.
Loop all the subtrees, use filter() to extract the sentence if it has the ADJ tag “JJ” and “SQ”(or “SBARQ”)
Store the sentence to a new list “ques”.



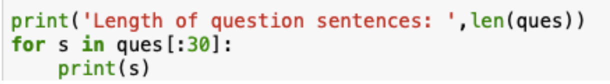
```
for sentence in tokensen:
    # Extract question sentences
    if sentence[len(sentence)-1] == '?':
        a = (nlp.parse(sentence))
        parsetree = Tree.fromstring(a)
        for subtree in parsetree.subtrees(filter =  filt):
            for subtree in parsetree.subtrees(filter =  f2):
                ques.append(sentence)
                break
            #ques.append(sentence)
            break
```

iv. For sentence does not satisfy the condition above, check if they end with exclamation mark, and store it to list “im” if so.

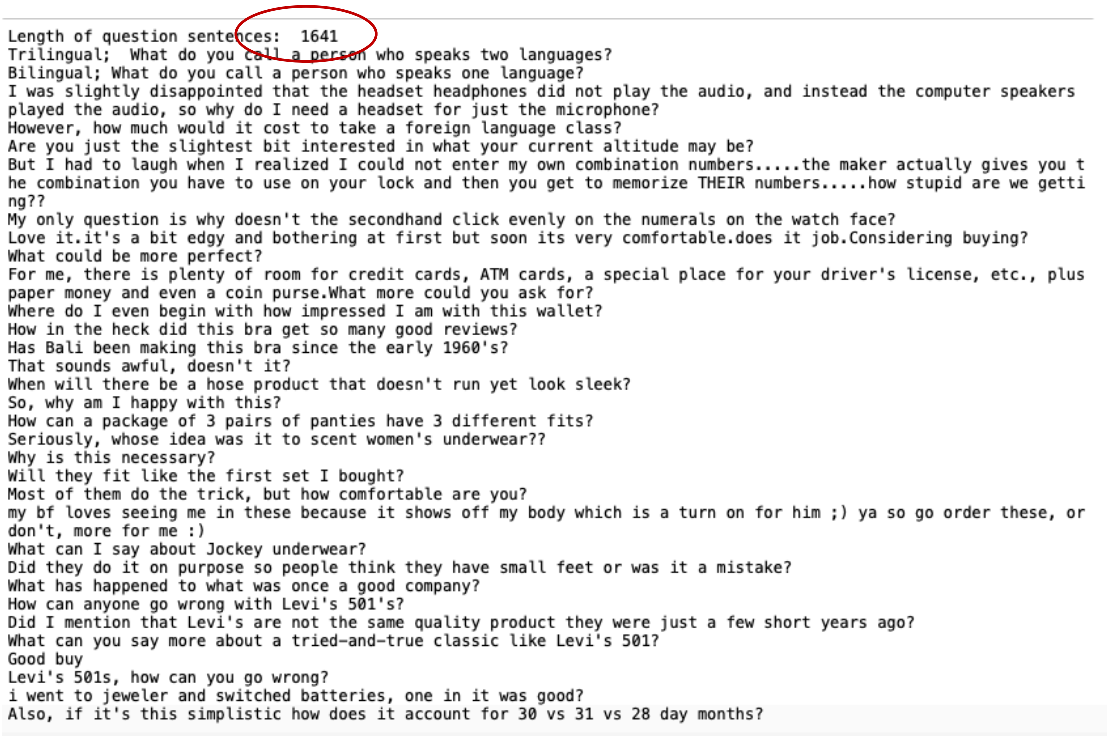


```
    elif sentence[len(sentence)-1] == '!':
        im.append(sentence)
```

d. Execution result



```
print('Length of question sentences: ',len(ques))
for s in ques[:30]:
    print(s)
```



```
Length of question sentences:  1641
Trilingual;  What do you call a person who speaks two languages?
Bilingual; What do you call a person who speaks one language?
I was slightly disappointed that the headset headphones did not play the audio, and instead the computer speakers
played the audio, so why do I need a headset for just the microphone?
However, how much would it cost to take a foreign language class?
Are you just the slightest bit interested in what your current altitude may be?
But I had to laugh when I realized I could not enter my own combination numbers.....the maker actually gives you t
he combination you have to use on your lock and then you get to memorize THEIR numbers.....how stupid are we getti
ng??
My only question is why doesn't the secondhand click evenly on the numerals on the watch face?
Love it.it's a bit edgy and bothering at first but soon its very comfortable.does it job.Considering buying?
What could be more perfect?
For me, there is plenty of room for credit cards, ATM cards, a special place for your driver's license, etc., plus
paper money and even a coin purse.What more could you ask for?
Where do I even begin with how impressed I am with this wallet?
How in the heck did this bra get so many good reviews?
Has Bali been making this bra since the early 1960's?
That sounds awful, doesn't it?
When will there be a hose product that doesn't run yet look sleek?
So, why am I happy with this?
How can a package of 3 pairs of panties have 3 different fits?
Seriously, whose idea was it to scent women's underwear??
Why is this necessary?
Will they fit like the first set I bought?
Most of them do the trick, but how comfortable are you?
my bf loves seeing me in these because it shows off my body which is a turn on for him ;) ya so go order these, or
don't, more for me :)
What can I say about Jockey underwear?
Did they do it on purpose so people think they have small feet or was it a mistake?
What has happened to what was once a good company?
How can anyone go wrong with Levi's 501's?
Did I mention that Levi's are not the same quality product they were just a few short years ago?
What can you say more about a tried-and-true classic like Levi's 501?
Good buy
Levi's 501s, how can you go wrong?
i went to jeweler and switched batteries, one in it was good?
Also, if it's this simplistic how does it account for 30 vs 31 vs 28 day months?
```

4. Extract imperative sentences includes Adjective phrases

a. Structure of imperative sentence

We need to understand the grammar and form of an imperative sentence before applying certain rules. The link is the Englishclub website, which gives the definition and form of the imperative sentence.

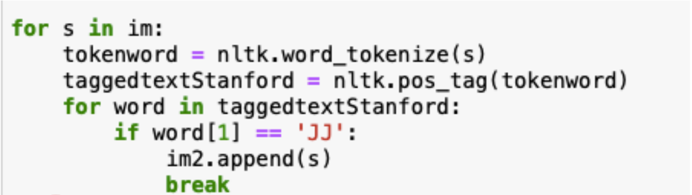
Usually an imperative sentence starts with a Verb type word.

<https://www.englishclub.com/grammar/sentence/type-imperative.htm>

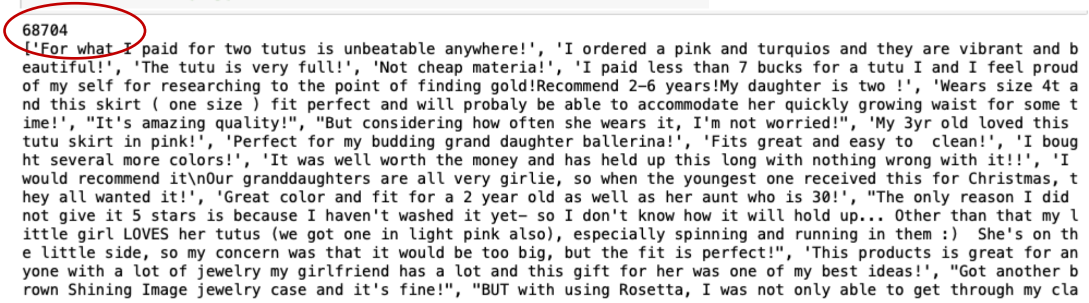
Note that we only consider the case with exclamation mark according to the assignment requirement.

b. Extract the imperative sentences

i. Tokenize the sentence and get the tag of each word, extract only the sentence has adjective phrase.



```
for s in im:
    tokenword = nltk.word_tokenize(s)
    taggedtextStanford = nltk.pos_tag(tokenword)
    for word in taggedtextStanford:
        if word[1] == 'JJ':
            im2.append(s)
            break
```

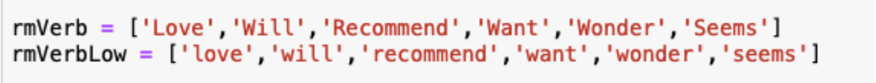


```
68704
['For what I paid for two tutus is unbeatable anywhere!', 'I ordered a pink and turquios and they are vibrant and b
eautiful!', 'The tutu is very full!', 'Not cheap materia!', 'I paid less than 7 bucks for a tutu I and I feel proud
of my self for researching to the point of finding gold!Recommend 2-6 years!My daughter is two !', 'Wears size 4t a
nd this skirt ( one size ) fit perfect and will probaly be able to accommodate her quickly growing waist for some t
ime!', "It's amazing quality!", "But considering how often she wears it, I'm not worried!", 'My 3yr old loved this
tutu skirt in pink!', 'Perfect for my budding grand daughter ballerina!', 'Fits great and easy to  clean!', 'I boug
ht several more colors!', 'It was well worth the money and has held up this long with nothing wrong with it!!', 'I
would recommend it\nOur granddaughters are all very girlie, so when the youngest one received this for Christmas, t
hey all wanted it!', 'Great color and fit for a 2 year old as well as her aunt who is 30!', "The only reason I did
not give it 5 stars is because I haven't washed it yet- so I don't know how it will hold up... Other than that my l
ittle girl LOVES her tutus (we got one in light pink also), especially spinning and running in them :)  She's on th
e little side, so my concern was that it would be too big, but the fit is perfect!", 'This products is great for an
yone with a lot of jewelry my girlfriend has a lot and this gift for her was one of my best ideas!', "Got another b
rown Shining Image jewelry case and it's fine!", "BUT with using Rosetta, I was not only able to get through my cla
```

ii. By observation, there are some sentences have a similar form with imperative sentences but are declarative type.

For example, “Recommend this product, it’s so great!”, “Just want a glass of water!”, and “Love it!”

Therefore, I create a list contains several common verbs for excluding sentences with similar form, to increase the accuracy.

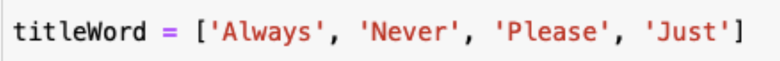


```
rmVerb = ['Love','Will','Recommend','Want','Wonder','Seems']
rmVerbLow = ['love','will','recommend','want','wonder','seems']
```

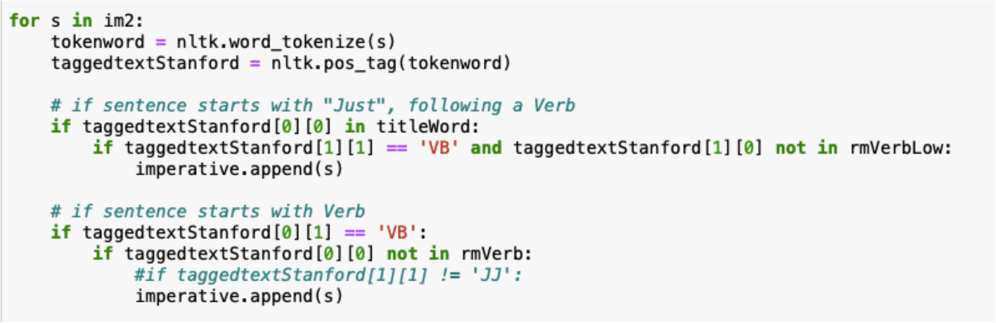
iii. Create the grammar rule and find the imperative sentences.

Besides the case the first word in sentence is Verb type, there are few more cases need to be considered, some imperative sentences start with words like “Please”, “Never”, etc..

For example, “Please be quite!” and “Always keep in mind!”.

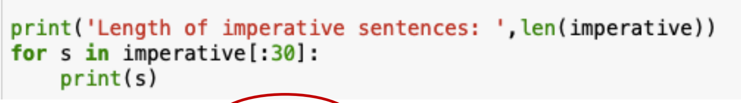


```
titleWord = ['Always', 'Never', 'Please', 'Just']
```

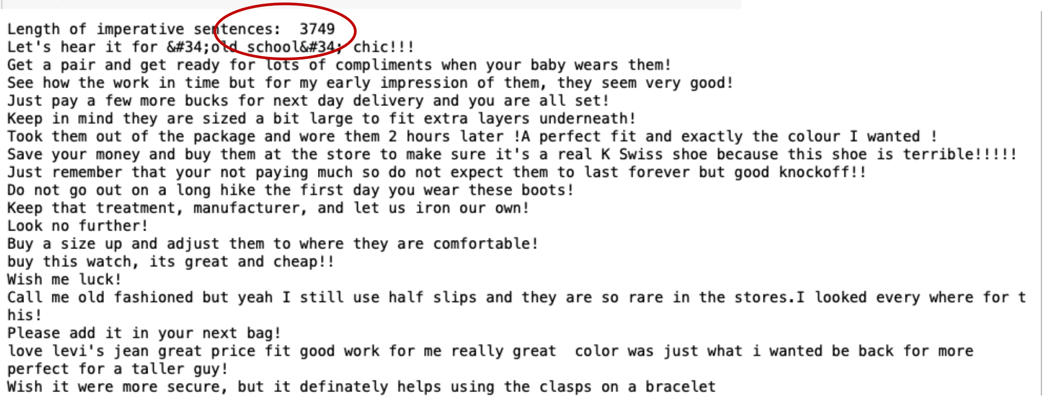


```
for s in im2:
    tokenword = nltk.word_tokenize(s)
    taggedtextStanford = nltk.pos_tag(tokenword)

    # if sentence starts with "Just", following a Verb
    if taggedtextStanford[0][0] in titleWord:
        if taggedtextStanford[1][1] == 'VB' and taggedtextStanford[1][0] not in rmVerbLow:
            imperative.append(s)

    # if sentence starts with Verb
    if taggedtextStanford[0][1] == 'VB':
        if taggedtextStanford[0][0] not in rmVerb:
            #if taggedtextStanford[1][1] != 'JJ':
            imperative.append(s)
```

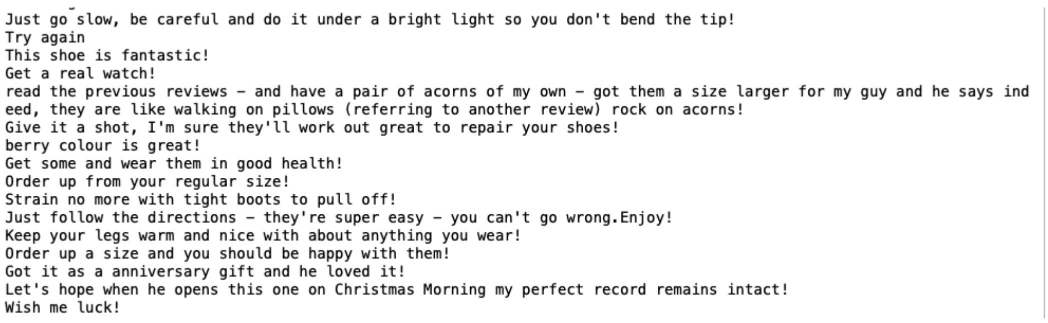
c. Execution result



```
print('Length of imperative sentences: ',len(imperative))
for s in imperative[:30]:
    print(s)
```



```
Length of imperative sentences:  3749
Let's hear it for &#34;old school&#34; chic!!!
Get a pair and get ready for lots of compliments when your baby wears them!
See how the work in time but for my early impression of them, they seem very good!
Just pay a few more bucks for next day delivery and you are all set!
Keep in mind they are sized a bit large to fit extra layers underneath!
Took them out of the package and wore them 2 hours later !A perfect fit and exactly the colour I wanted !
Save your money and buy them at the store to make sure it's a real K Swiss shoe because this shoe is terrible!!!!!
Just remember that your not paying much so do not expect them to last forever but good knockoff!!
Do not go out on a long hike the first day you wear these boots!
Keep that treatment, manufacturer, and let us iron our own!
Look no further!
Buy a size up and adjust them to where they are comfortable!
buy this watch, its great and cheap!!
Wish me luck!
Call me old fashioned but yeah I still use half slips and they are so rare in the stores.I looked every where for t
his!
Please add it in your next bag!
love levi's jean great price fit good work for me really great  color was just what i wanted be back for more
perfect for a taller guy!
Wish it were more secure, but it definately helps using the clasps on a bracelet
```

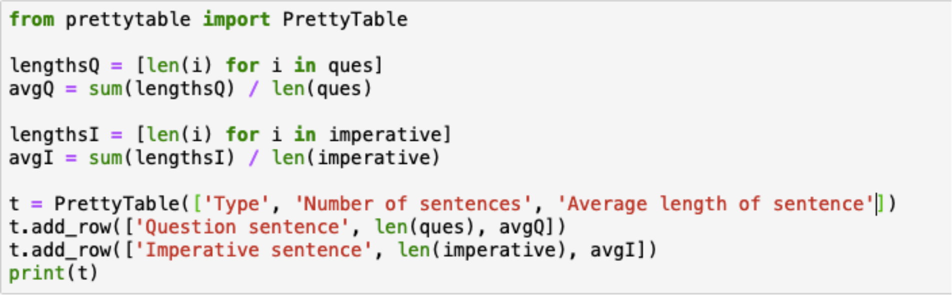


```
Just go slow, be careful and do it under a bright light so you don't bend the tip!
Try again
This shoe is fantastic!
Get a real watch!
read the previous reviews - and have a pair of acorns of my own - got them a size larger for my guy and he says ind
eed, they are like walking on pillows (referring to another review) rock on acorns!
Give it a shot, I'm sure they'll work out great to repair your shoes!
berry colour is great!
Get some and wear them in good health!
Order up from your regular size!
Strain no more with tight boots to pull off!
Just follow the directions - they're super easy - you can't go wrong.Enjoy!
Keep your legs warm and nice with about anything you wear!
Order up a size and you should be happy with them!
Got it as a anniversary gift and he loved it!
Let's hope when he opens this one on Christmas Morning my perfect record remains intact!
Wish me luck!
```

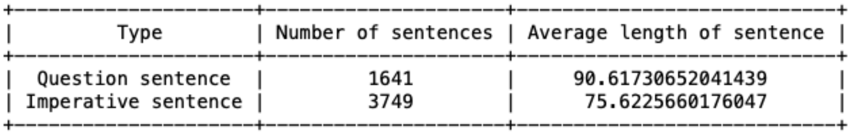
Analysis Process

1. **Summaries**

   Compute the number of sentences and average length of a sentence. The code and result are shown in figure below.



```
from prettytable import PrettyTable

lengthsQ = [len(i) for i in ques]
avgQ = sum(lengthsQ) / len(ques)

lengthsI = [len(i) for i in imperative]
avgI = sum(lengthsI) / len(imperative)

t = PrettyTable(['Type', 'Number of sentences', 'Average length of sentence'])
t.add_row(['Question sentence', len(ques), avgQ])
t.add_row(['Imperative sentence', len(imperative), avgI])
print(t)
```

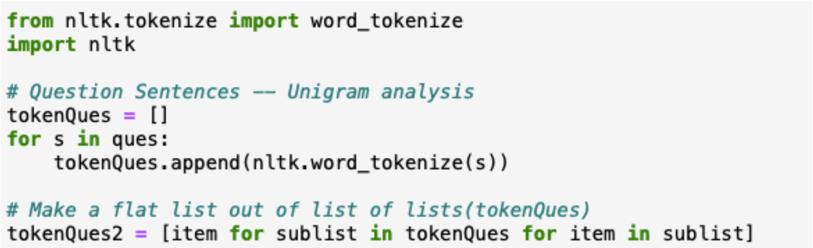


```
+---------------------+---------------------+----------------------------+
|         Type        | Number of sentences | Average length of sentence |
+---------------------+---------------------+----------------------------+
|  Question sentence  |         1641        |      90.61730652041439     |
| Imperative sentence |         3749        |      75.6225660176047      |
+---------------------+---------------------+----------------------------+
```

2. **Question sentences**

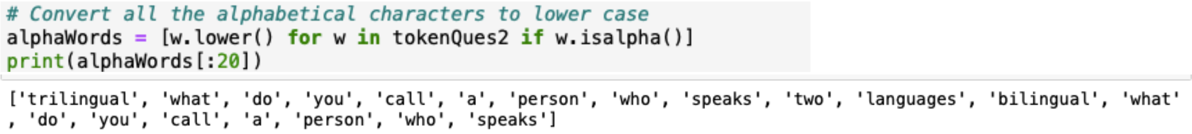
   a. Unigram frequency

      i. Tokenize the words in the list of question sentence



```
from nltk.tokenize import word_tokenize
import nltk

# Question Sentences -- Unigram analysis
tokenQues = []
for s in ques:
    tokenQues.append(nltk.word_tokenize(s))

# Make a flat list out of list of lists(tokenQues)
tokenQues2 = [item for sublist in tokenQues for item in sublist]
```

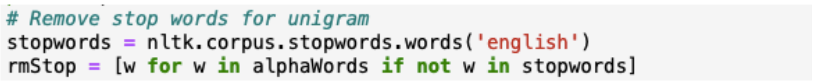
      ii. Lowercase, isalpha(). Convert all the alphabetical characters to lower case.



```
# Convert all the alphabetical characters to lower case
alphaWords = [w.lower() for w in tokenQues2 if w.isalpha()]
print(alphaWords[:20])
```

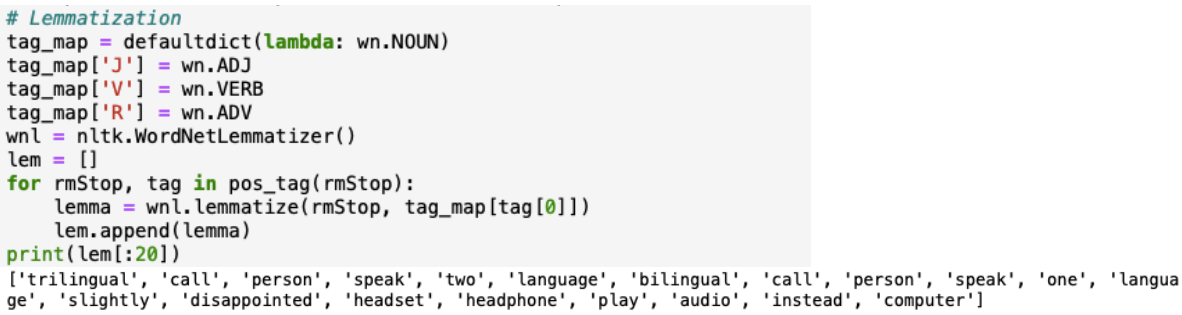
```
['trilingual', 'what', 'do', 'you', 'call', 'a', 'person', 'who', 'speaks', 'two', 'languages', 'bilingual', 'what'
, 'do', 'you', 'call', 'a', 'person', 'who', 'speaks']
```

      iii. Remove all the stop words.



```
# Remove stop words for unigram
stopwords = nltk.corpus.stopwords.words('english')
rmStop = [w for w in alphaWords if not w in stopwords]
```

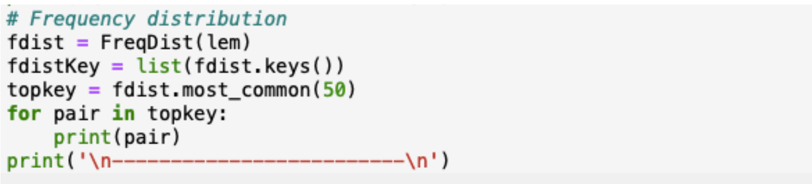
      iv. Lemmatization.



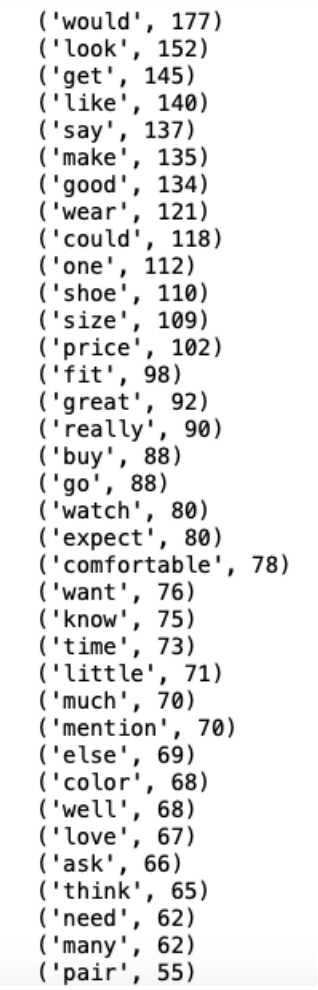
```
# Lemmatization
tag_map = defaultdict(lambda: wn.NOUN)
tag_map['J'] = wn.ADJ
tag_map['V'] = wn.VERB
tag_map['R'] = wn.ADV
wnl = nltk.WordNetLemmatizer()
lem = []
for rmStop, tag in pos_tag(rmStop):
    lemma = wnl.lemmatize(rmStop, tag_map[tag[0]])
    lem.append(lemma)
print(lem[:20])
```

```
['trilingual', 'call', 'person', 'speak', 'two', 'language', 'bilingual', 'call', 'person', 'speak', 'one', 'langua
ge', 'slightly', 'disappointed', 'headset', 'headphone', 'play', 'audio', 'instead', 'computer']
```

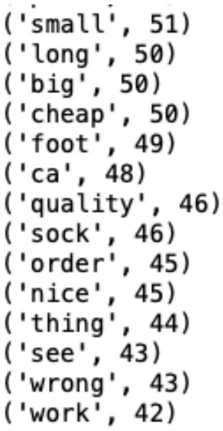
      v. Get the frequency



```python
# Frequency distribution
fdist = FreqDist(lem)
fdistKey = list(fdist.keys())
topkey = fdist.most_common(50)
for pair in topkey:
    print(pair)
print('\n----------------------\n')
```



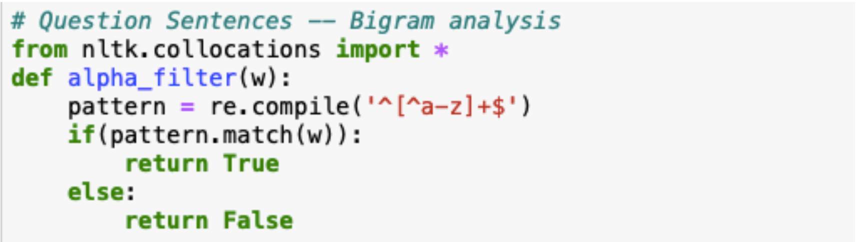
```
('would', 177)
('look', 152)
('get', 145)
('like', 140)
('say', 137)
('make', 135)
('good', 134)
('wear', 121)
('could', 118)
('one', 112)
('shoe', 110)
('size', 109)
('price', 102)
('fit', 98)
('great', 92)
('really', 90)
('buy', 88)
('go', 88)
('watch', 80)
('expect', 80)
('comfortable', 78)
('want', 76)
('know', 75)
('time', 73)
('little', 71)
('much', 70)
('mention', 70)
('else', 69)
('color', 68)
('well', 68)
('love', 67)
('ask', 66)
('think', 65)
('need', 62)
('many', 62)
('pair', 55)
```



```
('small', 51)
('long', 50)
('big', 50)
('cheap', 50)
('foot', 49)
('ca', 48)
('quality', 46)
('sock', 46)
('order', 45)
('nice', 45)
('thing', 44)
('see', 43)
('wrong', 43)
('work', 42)
```

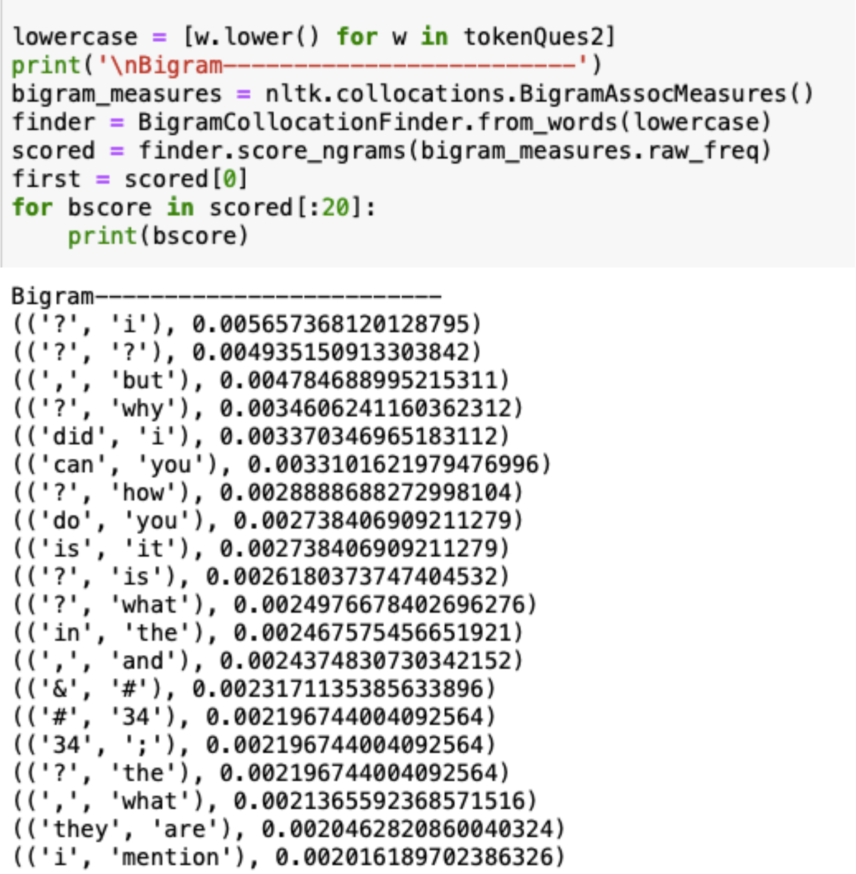
b. Bigram frequency

   i. Define a new function named *alpha_filter* which returns True if the character is non-alphabetical and False otherwise.



```python
# Question Sentences -- Bigram analysis
from nltk.collocations import *
def alpha_filter(w):
    pattern = re.compile('^[^a-z]+$')
    if(pattern.match(w)):
        return True
    else:
        return False
```

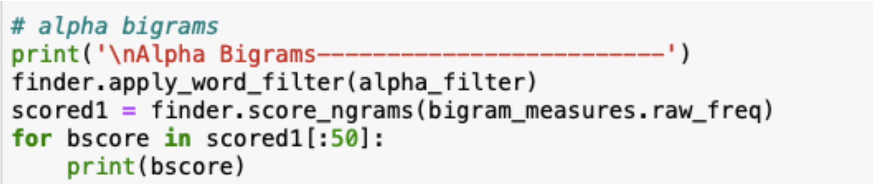
   ii. Define a variable named BigramCollocationFinder and import the collocation finder module.



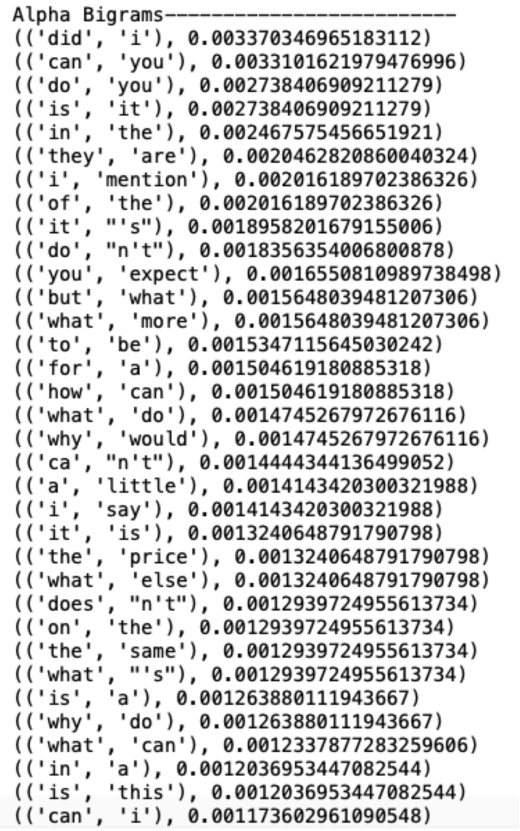
```
lowercase = [w.lower() for w in tokenQues2]
print('\nBigram-------------------------')
bigram_measures = nltk.collocations.BigramAssocMeasures()
finder = BigramCollocationFinder.from_words(lowercase)
scored = finder.score_ngrams(bigram_measures.raw_freq)
first = scored[0]
for bscore in scored[:20]:
    print(bscore)
```

```
Bigram-------------------------
(('?', 'i'), 0.005657368120128795)
(('?', '?'), 0.004935150913303842)
((',', 'but'), 0.004784688995215311)
(('?', 'why'), 0.0034606241160362312)
(('did', 'i'), 0.003370346965183112)
(('can', 'you'), 0.0033101621979476996)
(('?', 'how'), 0.0028888688272998104)
(('do', 'you'), 0.002738406909211279)
(('is', 'it'), 0.002738406909211279)
(('?', 'is'), 0.0026180373747404532)
(('?', 'what'), 0.0024976678402696276)
(('in', 'the'), 0.002467575456651921)
((',', 'and'), 0.0024374830730342152)
(('&', '#'), 0.0023171135385633896)
(('#', '34'), 0.002196744004092564)
(('34', ';'), 0.002196744004092564)
(('?', 'the'), 0.002196744004092564)
((',', 'what'), 0.0021365592368571516)
(('they', 'are'), 0.0020462820860040324)
(('i', 'mention'), 0.002016189702386326)
```

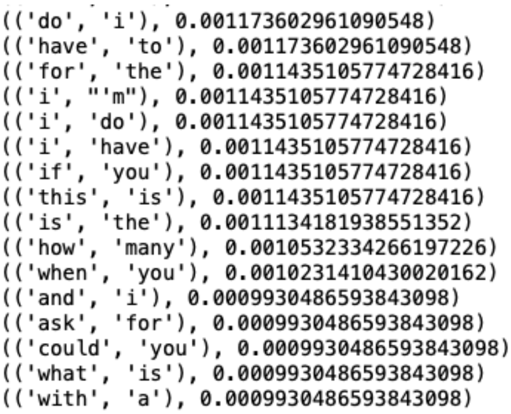
iii. Remove the non-analphabetic characters



```
# alpha bigrams
print('\nAlpha Bigrams-------------------------')
finder.apply_word_filter(alpha_filter)
scored1 = finder.score_ngrams(bigram_measures.raw_freq)
for bscore in scored1[:50]:
    print(bscore)
```

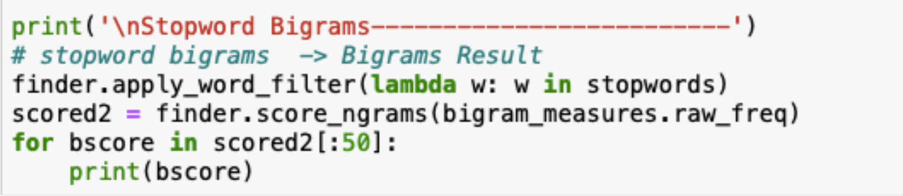


```
Alpha Bigrams-------------------------
(('did', 'i'), 0.003370346965183112)
(('can', 'you'), 0.0033101621979476996)
(('do', 'you'), 0.002738406909211279)
(('is', 'it'), 0.002738406909211279)
(('in', 'the'), 0.002467575456651921)
(('they', 'are'), 0.0020462820860040324)
(('i', 'mention'), 0.002016189702386326)
(('of', 'the'), 0.002016189702386326)
(('it', "'s"), 0.0018958201679155006)
(('do', "n't"), 0.0018356354006800878)
(('you', 'expect'), 0.0016550810989738498)
(('but', 'what'), 0.0015648039481207306)
(('what', 'more'), 0.0015648039481207306)
(('to', 'be'), 0.0015347115645030242)
(('for', 'a'), 0.001504619180885318)
(('how', 'can'), 0.001504619180885318)
(('what', 'do'), 0.0014745267972676116)
(('why', 'would'), 0.0014745267972676116)
(('ca', "n't"), 0.0014444344136499052)
(('a', 'little'), 0.0014143420300321988)
(('i', 'say'), 0.0014143420300321988)
(('it', 'is'), 0.0013240648791790798)
(('the', 'price'), 0.0013240648791790798)
(('what', 'else'), 0.0013240648791790798)
(('does', "n't"), 0.0012939724955613734)
(('on', 'the'), 0.0012939724955613734)
(('the', 'same'), 0.0012939724955613734)
(('what', "'s"), 0.0012939724955613734)
(('is', 'a'), 0.001263880111943667)
(('why', 'do'), 0.001263880111943667)
(('what', 'can'), 0.0012337877283259606)
(('in', 'a'), 0.0012036953447082544)
(('is', 'this'), 0.0012036953447082544)
(('can', 'i'), 0.001173602961090548)
```

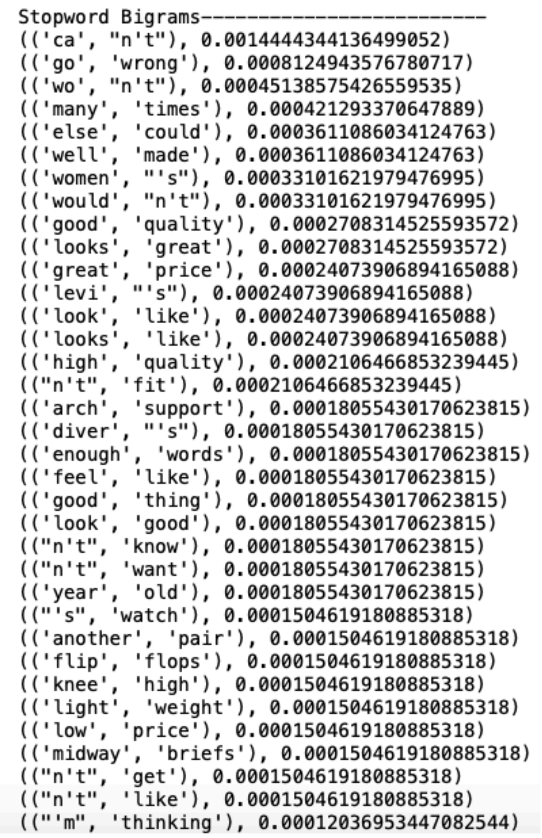


```
(('do', 'i'), 0.001173602961090548)
(('have', 'to'), 0.001173602961090548)
(('for', 'the'), 0.0011435105774728416)
(('i', "'m"), 0.0011435105774728416)
(('i', 'do'), 0.0011435105774728416)
(('i', 'have'), 0.0011435105774728416)
(('if', 'you'), 0.0011435105774728416)
(('this', 'is'), 0.0011435105774728416)
(('is', 'the'), 0.0011134181938551352)
(('how', 'many'), 0.0010532334266197226)
(('when', 'you'), 0.0010231410430020162)
(('and', 'i'), 0.0009930486593843098)
(('ask', 'for'), 0.0009930486593843098)
(('could', 'you'), 0.0009930486593843098)
(('what', 'is'), 0.0009930486593843098)
(('with', 'a'), 0.0009930486593843098)
```

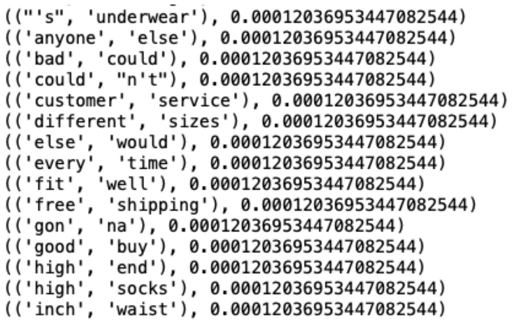
iv. Remove stopwords



```
print('\nStopword Bigrams-------------------------')
# stopword bigrams  -> Bigrams Result
finder.apply_word_filter(lambda w: w in stopwords)
scored2 = finder.score_ngrams(bigram_measures.raw_freq)
for bscore in scored2[:50]:
    print(bscore)
```



```
Stopword Bigrams-------------------------
(('ca', "n't"), 0.0014444344136499052)
(('go', 'wrong'), 0.0008124943576780717)
(('wo', "n't"), 0.00045138575426559535)
(('many', 'times'), 0.000421293370647889)
(('else', 'could'), 0.0003611086034124763)
(('well', 'made'), 0.0003611086034124763)
(('women', "'s"), 0.00033101621979476995)
(('would', "n't"), 0.00033101621979476995)
(('good', 'quality'), 0.0002708314525593572)
(('looks', 'great'), 0.0002708314525593572)
(('great', 'price'), 0.00024073906894165088)
(('levi', "'s"), 0.00024073906894165088)
(('look', 'like'), 0.00024073906894165088)
(('looks', 'like'), 0.00024073906894165088)
(('high', 'quality'), 0.0002106466853239445)
(("n't", 'fit'), 0.0002106466853239445)
(('arch', 'support'), 0.00018055430170623815)
(('diver', "'s"), 0.00018055430170623815)
(('enough', 'words'), 0.00018055430170623815)
(('feel', 'like'), 0.00018055430170623815)
(('good', 'thing'), 0.00018055430170623815)
(('look', 'good'), 0.00018055430170623815)
(("n't", 'know'), 0.00018055430170623815)
(("n't", 'want'), 0.00018055430170623815)
(('year', 'old'), 0.00018055430170623815)
(("'s", 'watch'), 0.0001504619180885318)
(('another', 'pair'), 0.0001504619180885318)
(('flip', 'flops'), 0.0001504619180885318)
(('knee', 'high'), 0.0001504619180885318)
(('light', 'weight'), 0.0001504619180885318)
(('low', 'price'), 0.0001504619180885318)
(('midway', 'briefs'), 0.0001504619180885318)
(("n't", 'get'), 0.0001504619180885318)
(("n't", 'like'), 0.0001504619180885318)
(("'m", 'thinking'), 0.00012036953447082544)
```

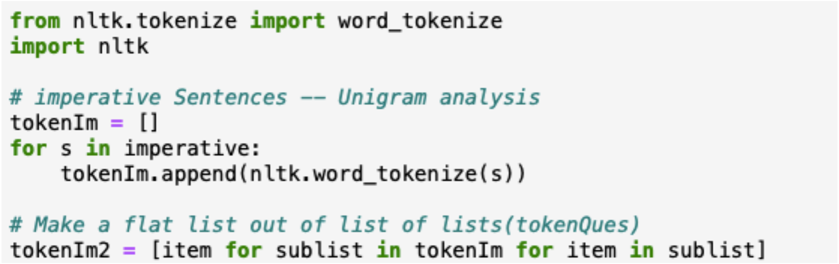


```
(("'s", 'underwear'), 0.00012036953447082544)
(('anyone', 'else'), 0.00012036953447082544)
(('bad', 'could'), 0.00012036953447082544)
(('could', "n't"), 0.00012036953447082544)
(('customer', 'service'), 0.00012036953447082544)
(('different', 'sizes'), 0.00012036953447082544)
(('else', 'would'), 0.00012036953447082544)
(('every', 'time'), 0.00012036953447082544)
(('fit', 'well'), 0.00012036953447082544)
(('free', 'shipping'), 0.00012036953447082544)
(('gon', 'na'), 0.00012036953447082544)
(('good', 'buy'), 0.00012036953447082544)
(('high', 'end'), 0.00012036953447082544)
(('high', 'socks'), 0.00012036953447082544)
(('inch', 'waist'), 0.00012036953447082544)
```

3. Imperative sentences

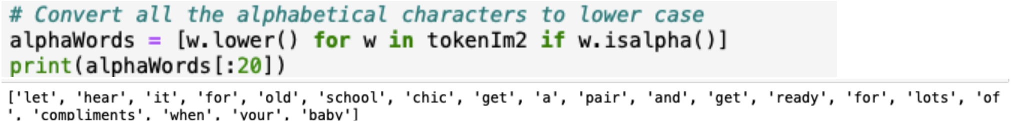
a. Unigram frequency

i. Tokenize the words in the list of question sentence



```
from nltk.tokenize import word_tokenize
import nltk

# imperative Sentences -- Unigram analysis
tokenIm = []
for s in imperative:
    tokenIm.append(nltk.word_tokenize(s))

# Make a flat list out of list of lists(tokenQues)
tokenIm2 = [item for sublist in tokenIm for item in sublist]
```

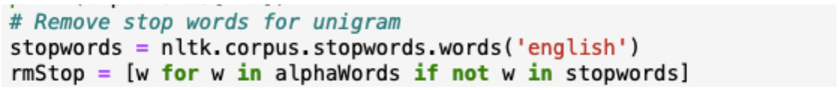
ii. Lowercase, isalpha(). Convert all the alphabetical characters to lower case.



```
# Convert all the alphabetical characters to lower case
alphaWords = [w.lower() for w in tokenIm2 if w.isalpha()]
print(alphaWords[:20])
```

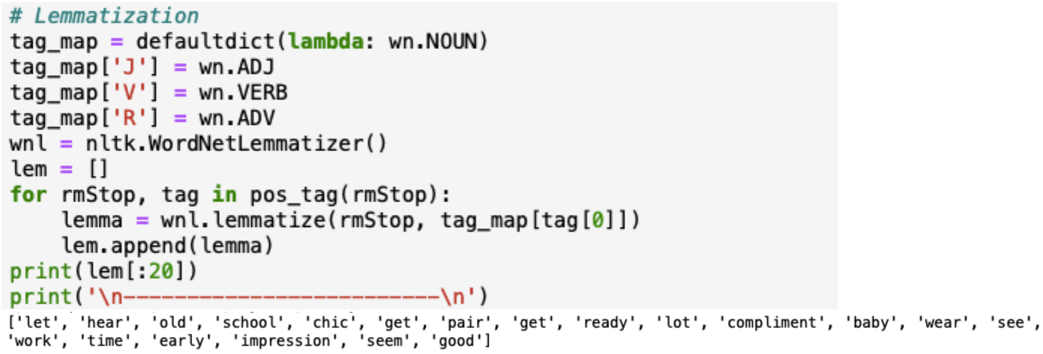
```
['let', 'hear', 'it', 'for', 'old', 'school', 'chic', 'get', 'a', 'pair', 'and', 'get', 'ready', 'for', 'lots', 'of
', 'compliments', 'when', 'your', 'baby']
```

iii. Remove all the stop words.



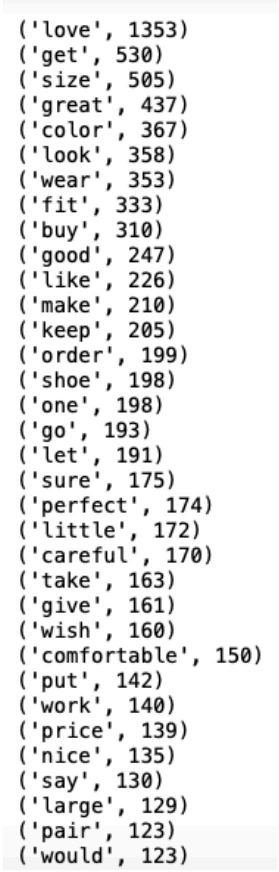
```
# Remove stop words for unigram
stopwords = nltk.corpus.stopwords.words('english')
rmStop = [w for w in alphaWords if not w in stopwords]
```

iv. Lemmatization.

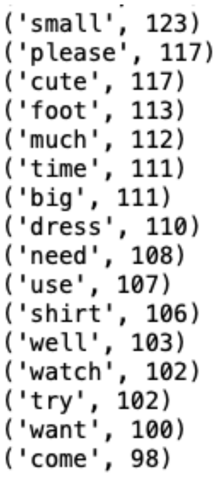


```
# Lemmatization
tag_map = defaultdict(lambda: wn.NOUN)
tag_map['J'] = wn.ADJ
tag_map['V'] = wn.VERB
tag_map['R'] = wn.ADV
wnl = nltk.WordNetLemmatizer()
lem = []
for rmStop, tag in pos_tag(rmStop):
    lemma = wnl.lemmatize(rmStop, tag_map[tag[0]])
    lem.append(lemma)
print(lem[:20])
print('\n------------------------\n')
['let', 'hear', 'old', 'school', 'chic', 'get', 'pair', 'get', 'ready', 'lot', 'compliment', 'baby', 'wear', 'see',
'work', 'time', 'early', 'impression', 'seem', 'good']
```

v. Get the frequency



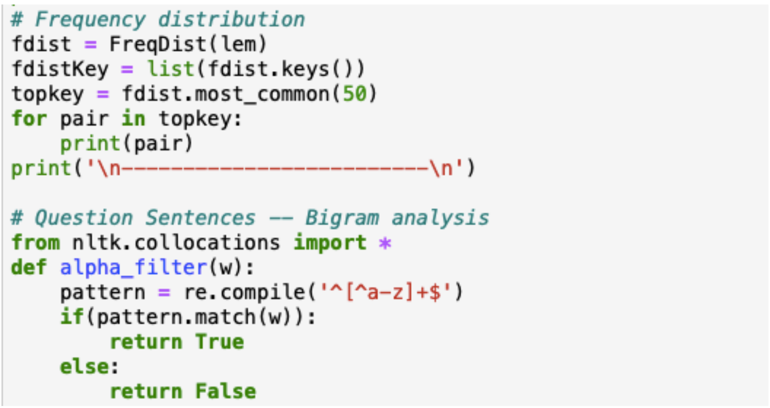
```
('love', 1353)
('get', 530)
('size', 505)
('great', 437)
('color', 367)
('look', 358)
('wear', 353)
('fit', 333)
('buy', 310)
('good', 247)
('like', 226)
('make', 210)
('keep', 205)
('order', 199)
('shoe', 198)
('one', 198)
('go', 193)
('let', 191)
('sure', 175)
('perfect', 174)
('little', 172)
('careful', 170)
('take', 163)
('give', 161)
('wish', 160)
('comfortable', 150)
('put', 142)
('work', 140)
('price', 139)
('nice', 135)
('say', 130)
('large', 129)
('pair', 123)
('would', 123)
```



```
('small', 123)
('please', 117)
('cute', 117)
('foot', 113)
('much', 112)
('time', 111)
('big', 111)
('dress', 110)
('need', 108)
('use', 107)
('shirt', 106)
('well', 103)
('watch', 102)
('try', 102)
('want', 100)
('come', 98)
```

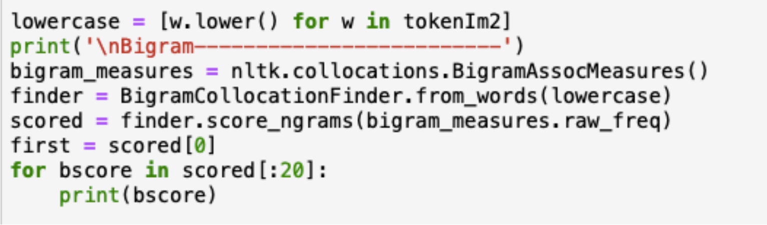
b. Bigram frequency

i. Define a new function named *alpha_filter* which returns True if the character is non-alphabetical and False otherwise.

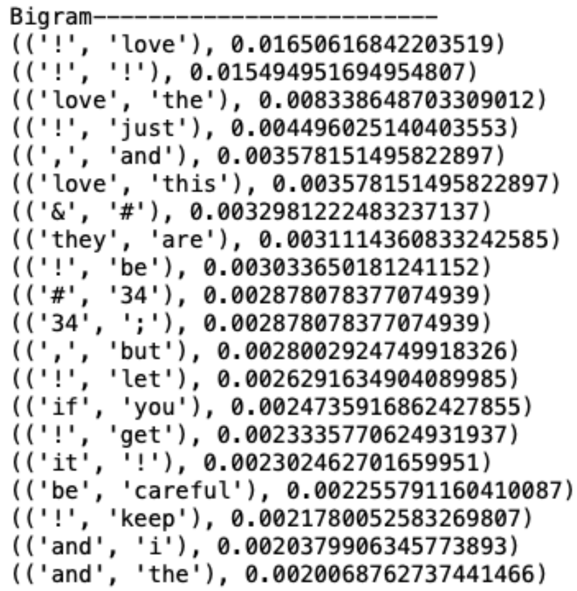


```
# Frequency distribution
fdist = FreqDist(lem)
fdistKey = list(fdist.keys())
topkey = fdist.most_common(50)
for pair in topkey:
    print(pair)
print('\n------------------------\n')

# Question Sentences -- Bigram analysis
from nltk.collocations import *
def alpha_filter(w):
    pattern = re.compile('^[^a-z]+$')
    if(pattern.match(w)):
        return True
    else:
        return False
```

ii. Define a variable named BigramCollocationFinder and import the collocation finder module.

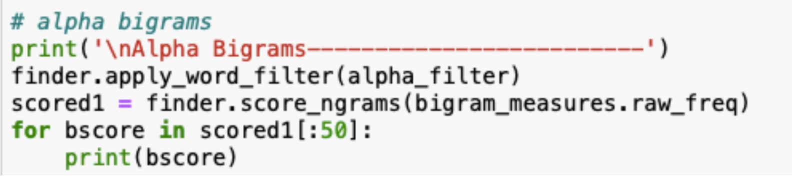


```
lowercase = [w.lower() for w in tokenIm2]
print('\nBigram------------------------')
bigram_measures = nltk.collocations.BigramAssocMeasures()
finder = BigramCollocationFinder.from_words(lowercase)
scored = finder.score_ngrams(bigram_measures.raw_freq)
first = scored[0]
for bscore in scored[:20]:
    print(bscore)
```

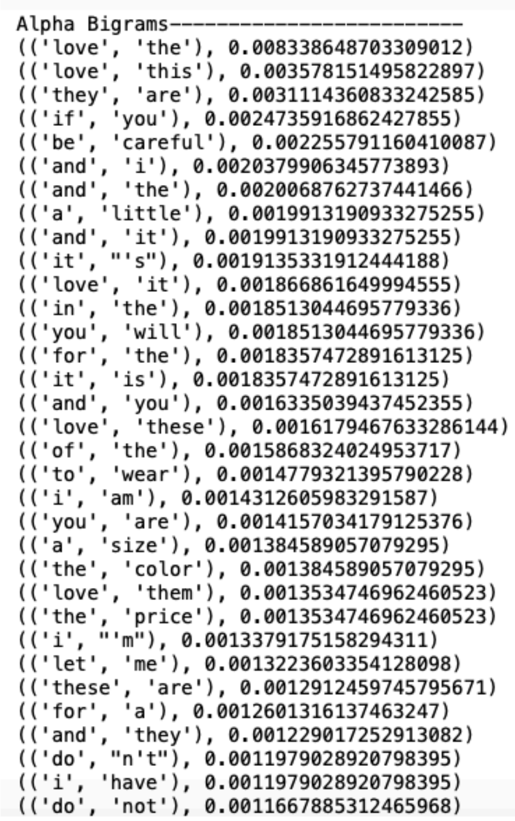


```
Bigram-------------------------
(('!', 'love'), 0.01650616842203519)
(('!', '!'), 0.015494951694954807)
(('love', 'the'), 0.008338648703309012)
(('!', 'just'), 0.004496025140403553)
((',', 'and'), 0.003578151495822897)
(('love', 'this'), 0.003578151495822897)
(('&', '#'), 0.0032981222483237137)
(('they', 'are'), 0.0031114360833242585)
(('!', 'be'), 0.003033650181241152)
(('#', '34'), 0.002878078377074939)
(('34', ';'), 0.002878078377074939)
((',', 'but'), 0.0028002924749918326)
(('!', 'let'), 0.002629163490408998 5)
(('if', 'you'), 0.0024735916862427855)
(('!', 'get'), 0.0023335770624931937)
(('it', '!'), 0.002302462701659951)
(('be', 'careful'), 0.002255791160410087)
(('!', 'keep'), 0.0021780052583269807)
(('and', 'i'), 0.0020379906345773893)
(('and', 'the'), 0.0020068762737441466)
```

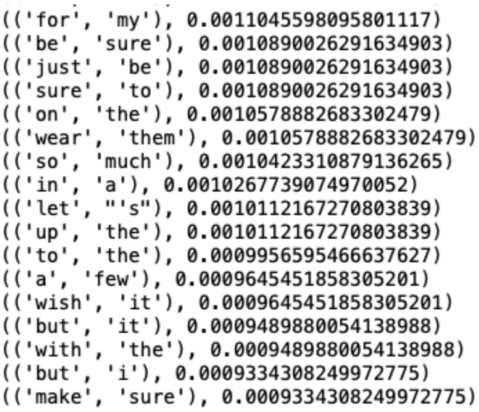
iii. Remove the non-analphabetic characters



```
# alpha bigrams
print('\nAlpha Bigrams-------------------------')
finder.apply_word_filter(alpha_filter)
scored1 = finder.score_ngrams(bigram_measures.raw_freq)
for bscore in scored1[:50]:
    print(bscore)
```

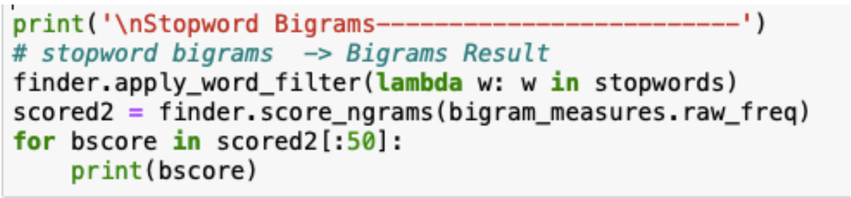


```
Alpha Bigrams--------------------------
(('love', 'the'), 0.008338648703309012)
(('love', 'this'), 0.003578151495822897)
(('they', 'are'), 0.0031114360833242585)
(('if', 'you'), 0.0024735916862427855)
(('be', 'careful'), 0.002255791160410087)
(('and', 'i'), 0.0020379906345773893)
(('and', 'the'), 0.0020068762737441466)
(('a', 'little'), 0.0019913190933275255)
(('and', 'it'), 0.0019913190933275255)
(('it', "'s"), 0.0019135331912444188)
(('love', 'it'), 0.001866861649994555)
(('in', 'the'), 0.0018513044695779336)
(('you', 'will'), 0.0018513044695779336)
(('for', 'the'), 0.0018357472891613125)
(('it', 'is'), 0.0018357472891613125)
(('and', 'you'), 0.0016335039437452355)
(('love', 'these'), 0.0016179467633286144)
(('of', 'the'), 0.0015868324024953717)
(('to', 'wear'), 0.0014779321395790228)
(('i', 'am'), 0.0014312605983291587)
(('you', 'are'), 0.0014157034179125376)
(('a', 'size'), 0.001384589057079295)
(('the', 'color'), 0.001384589057079295)
(('love', 'them'), 0.0013534746962460523)
(('the', 'price'), 0.0013534746962460523)
(('i', "'m"), 0.0013379175158294311)
(('let', 'me'), 0.0013223603354128098)
(('these', 'are'), 0.0012912459745795671)
(('for', 'a'), 0.0012601316137463247)
(('and', 'they'), 0.001229017252913082)
(('do', "n't"), 0.0011979028920798395)
(('i', 'have'), 0.0011979028920798395)
(('do', 'not'), 0.0011667885312465968)
```

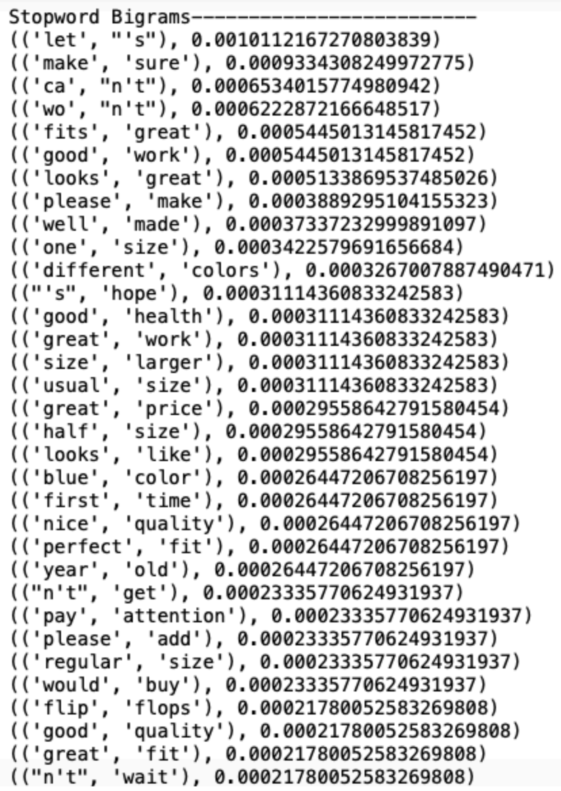


```
(('for', 'my'), 0.0011045598095801117)
(('be', 'sure'), 0.0010890026291634903)
(('just', 'be'), 0.0010890026291634903)
(('sure', 'to'), 0.0010890026291634903)
(('on', 'the'), 0.0010578882683302479)
(('wear', 'them'), 0.0010578882683302479)
(('so', 'much'), 0.0010423310879136265)
(('in', 'a'), 0.0010267739074970052)
(('let', "'s"), 0.0010112167270803839)
(('up', 'the'), 0.0010112167270803839)
(('to', 'the'), 0.0009956595466637627)
(('a', 'few'), 0.0009645451858305201)
(('wish', 'it'), 0.0009645451858305201)
(('but', 'it'), 0.0009489880054138988)
(('with', 'the'), 0.0009489880054138988)
(('but', 'i'), 0.0009334308249972775)
(('make', 'sure'), 0.0009334308249972775)
```

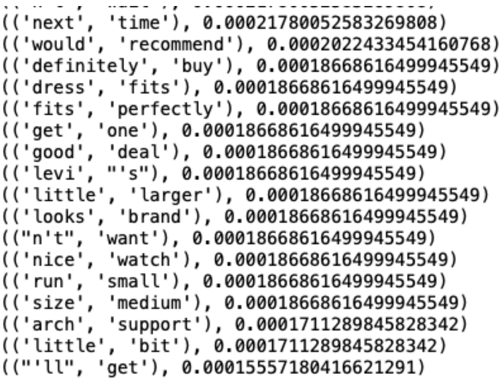
v. Remove stopwords



```
print('\nStopword Bigrams--------------------------')
# stopword bigrams  -> Bigrams Result
finder.apply_word_filter(lambda w: w in stopwords)
scored2 = finder.score_ngrams(bigram_measures.raw_freq)
for bscore in scored2[:50]:
    print(bscore)
```



```
Stopword Bigrams--------------------------
(('let', "'s"), 0.0010112167270803839)
(('make', 'sure'), 0.0009334308249972775)
(('ca', "n't"), 0.0006534015774980942)
(('wo', "n't"), 0.0006222872166648517)
(('fits', 'great'), 0.0005445013145817452)
(('good', 'work'), 0.0005445013145817452)
(('looks', 'great'), 0.0005133869537485026)
(('please', 'make'), 0.0003889295104155323)
(('well', 'made'), 0.0003733723299891097)
(('one', 'size'), 0.0003422579691656684)
(('different', 'colors'), 0.0003267007887490471)
(("'s", 'hope'), 0.00031114360833242583)
(('good', 'health'), 0.00031114360833242583)
(('great', 'work'), 0.00031114360833242583)
(('size', 'larger'), 0.00031114360833242583)
(('usual', 'size'), 0.00031114360833242583)
(('great', 'price'), 0.00029558642791580454)
(('half', 'size'), 0.00029558642791580454)
(('looks', 'like'), 0.00029558642791580454)
(('blue', 'color'), 0.00026447206708256197)
(('first', 'time'), 0.00026447206708256197)
(('nice', 'quality'), 0.00026447206708256197)
(('perfect', 'fit'), 0.00026447206708256197)
(('year', 'old'), 0.00026447206708256197)
(("n't", 'get'), 0.00023335770624931937)
(('pay', 'attention'), 0.00023335770624931937)
(('please', 'add'), 0.00023335770624931937)
(('regular', 'size'), 0.00023335770624931937)
(('would', 'buy'), 0.00023335770624931937)
(('flip', 'flops'), 0.00021780052583269808)
(('good', 'quality'), 0.00021780052583269808)
(('great', 'fit'), 0.00021780052583269808)
(("n't", 'wait'), 0.00021780052583269808)
```



```
(('next', 'time'), 0.00021780052583269808)
(('would', 'recommend'), 0.0002022433454160768)
(('definitely', 'buy'), 0.00018668616499945549)
(('dress', 'fits'), 0.00018668616499945549)
(('fits', 'perfectly'), 0.00018668616499945549)
(('get', 'one'), 0.00018668616499945549)
(('good', 'deal'), 0.00018668616499945549)
(('levi', "'s"), 0.00018668616499945549)
(('little', 'larger'), 0.00018668616499945549)
(('looks', 'brand'), 0.00018668616499945549)
(("n't", 'want'), 0.00018668616499945549)
(('nice', 'watch'), 0.00018668616499945549)
(('run', 'small'), 0.00018668616499945549)
(('size', 'medium'), 0.00018668616499945549)
(('arch', 'support'), 0.0001711289845828342)
(('little', 'bit'), 0.0001711289845828342)
(("'ll", 'get'), 0.00015557180416621291)
```

Interpretation

1. **Question sentences (with adjective phrase)**
   a. There are 1641 question sentences in total, simply calculate ratio: 1641/1140642 gives 0.144%. This indicates that only a few customers used question sentences when leaving a feedback.
   Extract the first few sentences from the list, we could see that most of them are questioning the seller about the quality and delivery. For example, wrong size or items sent to customer, or poor quality of the product. "How can anyone go wrong with Levi's 501's?", "How can a package of 3 pairs of panties have 3 different fits?"
   b. Analyzing the unigram frequency.
   Since this assignment is about sentiment, I tried to analyze the adjective words and most with high frequency are positive. For example, "good" (134), "great" (92), and "comfortable" (78). However, since this is question sentence, I do not think we could conclude that these reviews are positive even though the words they used are positive. It is possible that they did so to question the quality, like "are this supposed to be comfortable?".
   Many words with high frequency are about how feel about the products under clothing/shoes category. And several words are about size, such as "wear", "fit" and "size". This implies that most questions customers have are about clothes they bought, and probably they are not satisfied with the products.
   c. Analyzing the bigram frequency
   I chose the Alpha Bigrams that removes non-alphanumeric character first, to have a more accurate result of analysis. The bigrams with high frequency ("did I", "do you", "is it" etc.) are most likely not relevant to sentiment.
   For this reason, I decided to use the bigrams without stopwords. Most of the phrases (includes adjective word) with high frequency are positive, like "well made", "good quality", and "great price". It is reasonable to say that customers are satisfied with what they bought overall, based on this analysis result.
   Note that the top 2 bigram is "go wrong", which indicate that many customers might get the wrong items or the delivery was late.

2. **Imperative sentences (with adjective phrase)**
   a. There are 3749 imperative sentences in total, simply calculate ratio: 3749/1140642 gives 0.329%. Average length of the sentence is 75.62. This indicates that not many customers used imperative sentences when leaving feedback.
   Extract the first few sentences from the list, we could see that many of them are compliments: "Buy this watch, it's cheap and great!", "Please add it in your next bag!" We could conclude that most reviews are positive towards the products.
   b. Analyzing the unigram frequency.
   Similar with question sentence analysis, I focus on analyzing the adjective words. And most adjective words with high frequency are positive, like "great" (437), "good" (247), "perfect" (174).
   The unigram of the highest frequency is "love" (1353).
   It's reasonable to conclude that most customers left imperative sentence as reviews are satisfied and happy with the products.

c. Analyzing the bigram frequency

I chose the Alpha Bigrams that removes non-alphanumeric character first, to have a more accurate result of analysis. The top two bigrams are “love the” and “love this”, which implies that many customers have positive attitude towards the products.

However, many other bigrams with high frequency are likely not relevant to sentiment, like “they are”, “and I”, and “in the”.

For this reason, I decided to use the bigrams without stopwords. And most of the phrases (includes adjective word) with high frequency are positive. In fact, I could hardly find one that’s negative. For example, “good work”, “perfect fit”, “good health”.

Note that the top 8 bigram is “please make”, which might be customers asking the seller to produce more products. This suggest that Amazon could analyze this kind of reviews and use it to help making decision on the quantity of products produced.

Thoughts on conducting sentiment analysis

Python code

1. **What should be kept**
   a. Extracting the adjective phrases turns out to be necessary and essential for sentiment analysis, as we could see how customers feel about the products directly. It could help us cleaning many sentences that are not important. For example, simple declarative sentence like “I bought it last week.”
   b. Removing the stopwords seems necessary in some cases, as they are not relevant to the sentiment sometimes.
2. **Additional analysis tasks needed**
   a. Analysis on the adjective words is necessary in the future, especially the positive/negative analysis. We could make the positive and negative tags for all the adjective phrases and analyze these two sets of them. We could then analyze what are customers satisfied with and what are they unhappy with based on the sets.
   b. Some Verbs might need to be excluded in the future, as they are not relevant to the sentiment analysis and could affect the accuracy. For example, words like “get” and “buy” have high frequency but they are not really useful, we could not get much information from them.