NLP Assignment 2

Name: Bo Li

Date: 3/9//2020

Summary:

```
+---------------------+---------+--------------------------+---------------------+----------------------+
|         Type        | Numbers | Avg Length (Characters)  | Avg Length (Tokens) | Percentage in Total  |
+---------------------+---------+--------------------------+---------------------+----------------------+
|  Question Sentence  |   3682  |    136.81966322650734    |  30.422053231939163 | 0.4056960519008877%  |
| Imperative Sentence |   1127  |    90.41437444543034     |  20.35226264418811  | 0.12417692843354165% |
+---------------------+---------+--------------------------+---------------------+----------------------+
```

Unigram for Question: (30 Unigrams)

```
('look', 528)
('like', 522)
('size', 481)
('would', 463)
('get', 443)
('wear', 415)
('fit', 414)
('say', 401)
('make', 377)
('shoe', 376)
('one', 348)
('good', 343)
('well', 334)
('great', 323)
('could', 300)
('love', 273)
('really', 258)
('comfortable', 248)
('want', 248)
('watch', 242)
('go', 237)
('color', 235)
('know', 227)
('order', 226)
('small', 224)
('much', 219)
('little', 218)
('price', 215)
('pair', 209)
('right', 205)
```

Unigram for Imperative: (30 Unigrams)

```
('look', 329)
('great', 260)
('fit', 189)
('get', 139)
('love', 118)
('size', 118)
('comfortable', 116)
('like', 115)
('wear', 112)
('make', 109)
('good', 107)
('go', 99)
('nice', 94)
('would', 88)
('well', 85)
('shoe', 79)
('order', 77)
('cute', 75)
('buy', 62)
('perfect', 62)
('ca', 59)
('price', 59)
('sure', 58)
('really', 57)
('recommend', 55)
('color', 53)
('bought', 53)
('small', 53)
('shirt', 51)
('pair', 50)
```

Bigram for Question: (20 Bigrams)

```
(('go', 'wrong'), 0.00031246094238220225)
(('well', 'made'), 0.00028567857589229916)
(('year', 'old'), 0.00024104129840912743)
(('look', 'like'), 0.00023211384291249308)
(('good', 'quality'), 0.0001964040209259557)
(('looks', 'great'), 0.0001964040209259557)
(('would', 'recommend'), 0.00018747656542932134)
(('feel', 'like'), 0.000178549109932687)
(('another', 'pair'), 0.00016962165443605264)
(('high', 'quality'), 0.0001606941989394183)
(('make', 'sure'), 0.0001606941989394183)
(('highly', 'recommend'), 0.00015176674344278393)
(('fits', 'great'), 0.00014283928794614958)
(('little', 'bit'), 0.00014283928794614958)
(('looks', 'like'), 0.00014283928794614958)
(('much', 'better'), 0.00014283928794614958)
(('size', 'smaller'), 0.00013391183244951523)
(('years', 'ago'), 0.00013391183244951523)
(('look', 'good'), 0.00012498437695288088)
(('one', 'size'), 0.00012498437695288088)
```

Bigram for Imperative: (20 Bigrams)

```
(('looks', 'great'), 0.0034006190870645684)
(('fits', 'great'), 0.001307930418101757)
(('go', 'wrong'), 0.001264332737498365)
(('make', 'sure'), 0.0010463443344814055)
(('looks', 'good'), 0.0009155512926712299)
(('look', 'great'), 0.0006975628896542704)
(('would', 'recommend'), 0.0006103675284474866)
(('fits', 'perfect'), 0.000435976806033919)
(('well', 'made'), 0.000435976806033919)
(('would', 'buy'), 0.000435976806033919)
(('fit', 'great'), 0.0003923791254305271)
(('good', 'quality'), 0.0003923791254305271)
(('highly', 'recommend'), 0.0003923791254305271)
(('super', 'cute'), 0.0003923791254305271)
(('feels', 'great'), 0.0003487814448271352)
(('great', 'price'), 0.0003487814448271352)
(('say', 'enough'), 0.0003487814448271352)
(('fits', 'well'), 0.0003051837642237433)
(('look', 'good'), 0.0003051837642237433)
(('look', 'like'), 0.0003051837642237433)
```

Grammar Using:

To analyze both Question and Imperative sentences, I used **StanfordCoreNLP** to generate CFG and accomplish to analyze.
Both Question and Imperative sentences include "ADJP" in their CFG.

- To Extract a Question sentence: (Adj Phrase & Question sentences || "?" mark)
  - If it contains the symbol "?".
  - If "SBARQ" is included in a sentence, "SBARQ" indicates "WH" questions s.t What, Where, Which, Who, How, etc.
  - If "SQ" is included in a sentence, "SQ" indicates yes/no questions s.t "Do you …", "Can you…" and some other formats.
  - Exclude the sentence including symbol "!" because I don't count the sentence including it as a question sentence.
  -

- To Extract Imperative sentences (Adj Phrase & Imperative sentences & "!" mark)
  - If it contains the symbol "!".
  - Exclude the sentence including symbol "?" because we premiraly consider the sentence with "?" is a question sentence.
  - If "S" and "VP" in corresponding position of a sentence, and they both should be adjacent. Following by the structure given in the instructive.

Processing:

1. Split contents in "clothing_shoes_jewelry.txt" into lines using 'splitlines()'.

```
lines = contents.splitlines()
print(len(lines))
print(type(lines))
print(lines[:10])
```

```
2786318
<class 'list'>
['reviewerID:A1KLRMWW2FWPL4', 'asin:0000031887', 'reviewerName:Amazon Customer "cameramom"', 'helpful:[0, 0]', "revie
wText: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++", 'overall:5.0', 'summary:Great tutu-  not cheaply mad
e', 'unixReviewTime:1297468800', 'reviewTime:02 12, 2011', '']
```

2. Extract lines only including "reviewText" to retrieve all review texts.

```
# Remove title "reviewText:"
reviewText = [r[11:] for r in lines if "reviewText" in r ] # r[11:] is contents after 'reviewText:'
print(len(reviewText))
print(reviewText[:3])
```

```
278677
["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 thought it was adorable. I bought this to go with a light blue l
ong 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 f
or they the fuccia 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.']
```

3. Save all reviews into a file.

```
# save extracted reviews into file
saveFile = open('/Users/boli/Desktop/nlp/HW1/reviews.txt', 'w')

for r in reviewText:
    saveFile.write(r + '\n')

saveFile.close()
```

4. Convert list to string using “join”

```
str_join = ",".join(reviewText)

print(len(str_join))
print(str_join[:300])
```

```
88129771
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 fo
und such an affordable tutu that isn't made poorly. A++,I bought this for my 4 yr old daughter for dance class, she w
ore it today for the first time and the teacher thought it was ado
```

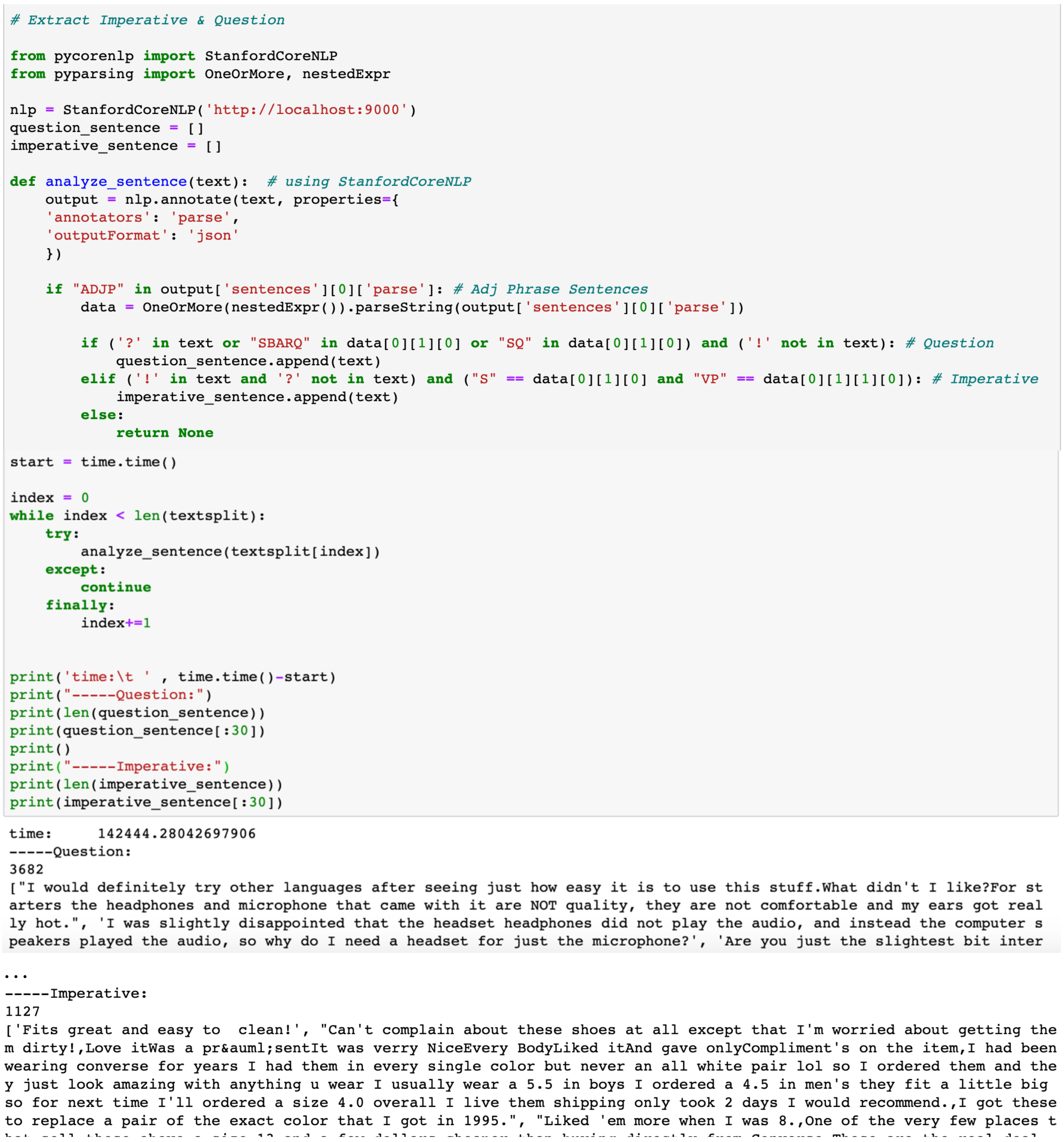
5. Make sentence tokenization to string using “nltk.sent_tokenize()”

```
# sentences tokenize
textsplit = nltk.sent_tokenize(str_join)

print(len(textsplit))
print(textsplit[:20])
```

```
907576
['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 Amazo
n and found such an affordable tutu that isn't made poorly.", 'A++,I bought this for my 4 yr old daughter for dance c
lass, she wore it today for the first time and the teacher thought it was adorable.', 'I bought this to go with a lig
```

6. ***Important**: Extract Imperative **Imperative** & **Question** Sentences based on the instruction given. I invoked “StanfordCoreNLP” API to analyze the CFG based on each sentence. Based on CFG in each sentence, I made the check to extract Imperative & Question sentences using if-statement conditions and assign them into each individual list. However, the inefficient here is to TAKE TOO LONG TIME. (142444 s)

   Note*. To setup environment of “StandfordCoreNLP”, see the link: https://github.com/dasmith/stanford-corenlp-python



```python
# Extract Imperative & Question

from pycorenlp import StanfordCoreNLP
from pyparsing import OneOrMore, nestedExpr

nlp = StanfordCoreNLP('http://localhost:9000')
question_sentence = []
imperative_sentence = []

def analyze_sentence(text):  # using StanfordCoreNLP
    output = nlp.annotate(text, properties={
    'annotators': 'parse',
    'outputFormat': 'json'
    })

    if "ADJP" in output['sentences'][0]['parse']: # Adj Phrase Sentences
        data = OneOrMore(nestedExpr()).parseString(output['sentences'][0]['parse'])

        if ('?' in text or "SBARQ" in data[0][1][0] or "SQ" in data[0][1][0]) and ('!' not in text): # Question
            question_sentence.append(text)
        elif ('!' in text and '?' not in text) and ("S" == data[0][1][0] and "VP" == data[0][1][1][0]): # Imperative
            imperative_sentence.append(text)
        else:
            return None
```

```python
start = time.time()

index = 0
while index < len(textsplit):
    try:
        analyze_sentence(textsplit[index])
    except:
        continue
    finally:
        index+=1


print('time:\t ' , time.time()-start)
print("-----Question:")
print(len(question_sentence))
print(question_sentence[:30])
print()
print("-----Imperative:")
print(len(imperative_sentence))
print(imperative_sentence[:30])
```

```
time:	 142444.28042697906
-----Question:
3682
["I would definitely try other languages after seeing just how easy it is to use this stuff.What didn't I like?For st
arters the headphones and microphone that came with it are NOT quality, they are not comfortable and my ears got real
ly hot.", 'I was slightly disappointed that the headset headphones did not play the audio, and instead the computer s
peakers played the audio, so why do I need a headset for just the microphone?', 'Are you just the slightest bit inter

...
-----Imperative:
1127
['Fits great and easy to  clean!', "Can't complain about these shoes at all except that I'm worried about getting the
m dirty!,Love itWas a pr&auml;sentIt was verry NiceEvery BodyLiked itAnd gave onlyCompliment's on the item,I had been
wearing converse for years I had them in every single color but never an all white pair lol so I ordered them and the
y just look amazing with anything u wear I usually wear a 5.5 in boys I ordered a 4.5 in men's they fit a little big
so for next time I'll ordered a size 4.0 overall I live them shipping only took 2 days I would recommend.,I got these
to replace a pair of the exact color that I got in 1995.", "Liked 'em more when I was 8.,One of the very few places t
```

7. Calculate Average length by **Characters** in both **Imperative** & **Question** sentences:

```
# Avg length of imperative sentence
imp_len = 0
for i in imperative_sentence:
    imp_len += len(i)

imp_avg_len = imp_len / len(imperative_sentence)

print(imp_avg_len)
```

```
90.41437444543034
```

```
# Avg length of question sentence
ques_len = 0
for i in question_sentence:
    ques_len += len(i)

ques_avg_len = ques_len / len(question_sentence)

print(ques_avg_len)
```

```
136.81966322650734
```

8. Calculate Average length by **Tokens** in both **Imperative** & **Question** sentences:

```
ques_token_len = 0
ques_tokens = [nltk.word_tokenize(t) for t in question_sentence]

for i in ques_tokens:
    ques_token_len += len(i)

ques_avg_token = ques_token_len / len(ques_tokens)

print(ques_avg_token)
```

```
30.422053231939163
```

```
imp_token_len = 0
imp_tokens = [nltk.word_tokenize(t) for t in imperative_sentence]

for i in imp_tokens:
    imp_token_len += len(i)

imp_avg_token = imp_token_len / len(imp_tokens)

print(imp_avg_token)
```

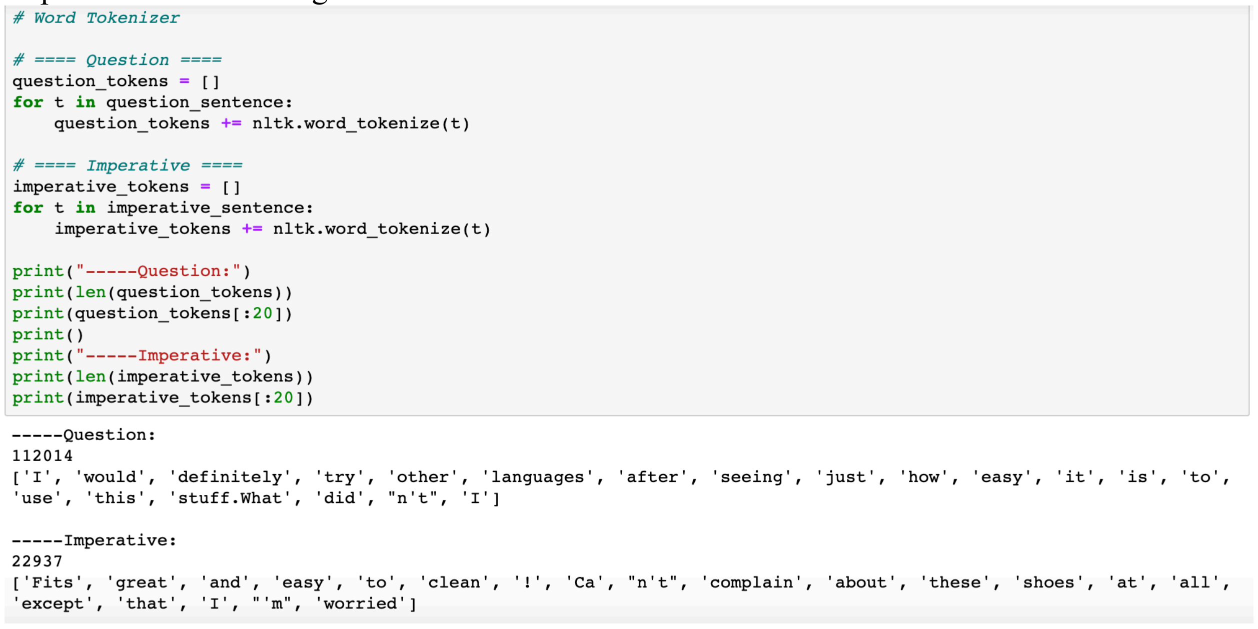
```
20.35226264418811
```

9. Create Table for above Statistics Data

```
# Create table to show data
from prettytable import PrettyTable
table = PrettyTable()
table.field_names = ["Type", "Numbers", "Avg Length (Characters)", "Avg Length (Tokens)", "Percentage in Total"]
table.add_row(["Question Sentence", len(question_sentence), ques_avg_len, ques_avg_token, str(len(question_sentence)/le
table.add_row(["Imperative Sentence", len(imperative_sentence), imp_avg_len, imp_avg_token, str(len(imperative_sentence

print(table)
```

```
+---------------------+---------+-------------------------+---------------------+----------------------+
|         Type        | Numbers | Avg Length (Characters) | Avg Length (Tokens) | Percentage in Total  |
+---------------------+---------+-------------------------+---------------------+----------------------+
|  Question Sentence  |   3682  |    136.81966322650734   |  30.422053231939163 | 0.40569605190088778% |
| Imperative Sentence |   1127  |    90.41437444543034    |  20.35226264418811  | 0.12417692843354165% |
+---------------------+---------+-------------------------+---------------------+----------------------+
```

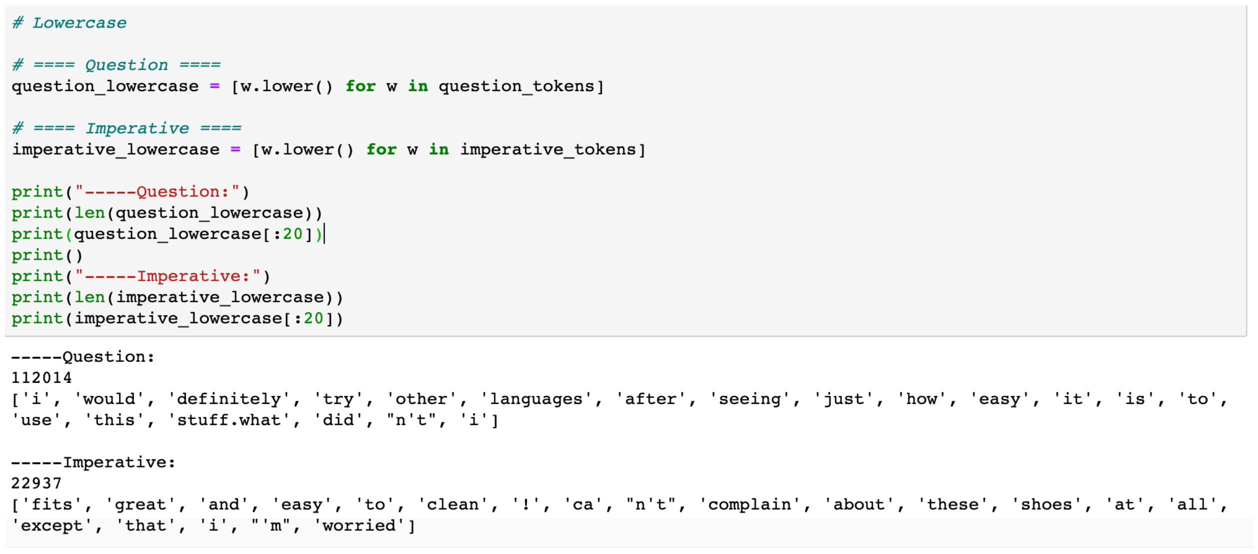
10. Using nltk.word_tokenize to tokenize both **Imperative** & **Question** sentences to get single token of each word.
    - nltk.word_tokenize could be used to split tokens based on white space and punctuation in a string.



```
# Word Tokenizer

# ==== Question ====
question_tokens = []
for t in question_sentence:
    question_tokens += nltk.word_tokenize(t)

# ==== Imperative ====
imperative_tokens = []
for t in imperative_sentence:
    imperative_tokens += nltk.word_tokenize(t)

print("-----Question:")
print(len(question_tokens))
print(question_tokens[:20])
print()
print("-----Imperative:")
print(len(imperative_tokens))
print(imperative_tokens[:20])
```

```
-----Question:
112014
['I', 'would', 'definitely', 'try', 'other', 'languages', 'after', 'seeing', 'just', 'how', 'easy', 'it', 'is', 'to',
'use', 'this', 'stuff.What', 'did', "n't", 'I']

-----Imperative:
22937
['Fits', 'great', 'and', 'easy', 'to', 'clean', '!', 'Ca', "n't", 'complain', 'about', 'these', 'shoes', 'at', 'all',
'except', 'that', 'I', "'m", 'worried']
```

11. Convert all uppercase characters to lowercase in tokens on both **Imperative** & **Question** sentences.
    - Using lowercase could be easily identified tokens without uppercase affect. e.g. This -> this, it's same as 'this', will be counted together then.



```
# Lowercase

# ==== Question ====
question_lowercase = [w.lower() for w in question_tokens]

# ==== Imperative ====
imperative_lowercase = [w.lower() for w in imperative_tokens]

print("-----Question:")
print(len(question_lowercase))
print(question_lowercase[:20])
print()
print("-----Imperative:")
print(len(imperative_lowercase))
print(imperative_lowercase[:20])
```

```
-----Question:
112014
['i', 'would', 'definitely', 'try', 'other', 'languages', 'after', 'seeing', 'just', 'how', 'easy', 'it', 'is', 'to',
'use', 'this', 'stuff.what', 'did', "n't", 'i']

-----Imperative:
22937
['fits', 'great', 'and', 'easy', 'to', 'clean', '!', 'ca', "n't", 'complain', 'about', 'these', 'shoes', 'at', 'all',
'except', 'that', 'i', "'m", 'worried']
```

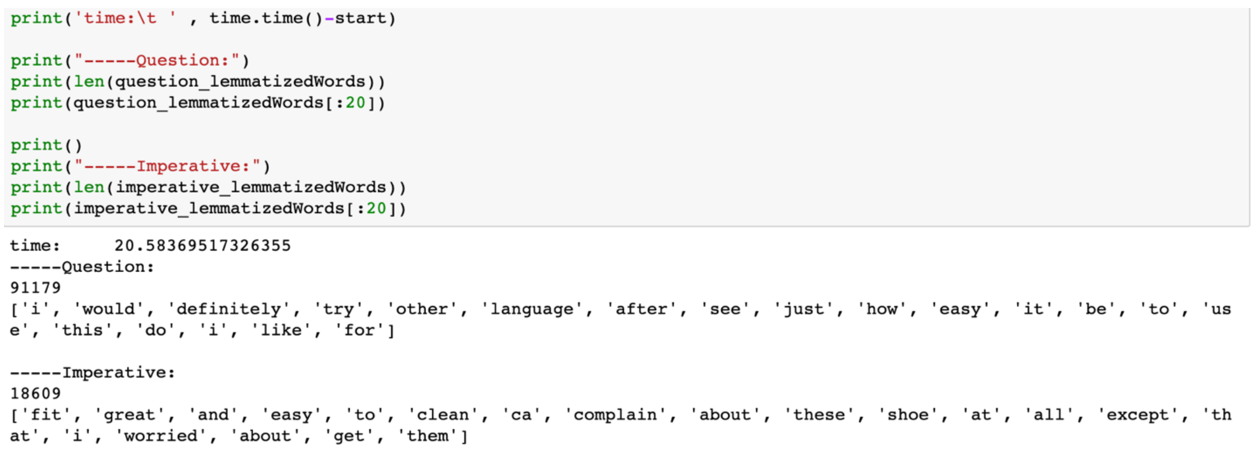
12. Using isalpha() to get all alphabet tokens only on both **Imperative** & **Question** sentences.
    - We only care about tokens with alphabet, no punctuations, other characters or numbers.

```python
# alphabets

# ==== Question ====
question_alphaWords = [w for w in question_lowercase if w.isalpha()]

# ==== Imperative ====
imperative_alphaWords = [w for w in imperative_lowercase if w.isalpha()]


print("-----Question:")
print(len(question_alphaWords))
print(question_alphaWords[:20])

print()
print("-----Imperative:")
print(len(imperative_alphaWords))
print(imperative_alphaWords[:20])
```

```
-----Question:
91179
['i', 'would', 'definitely', 'try', 'other', 'languages', 'after', 'seeing', 'just', 'how', 'easy', 'it', 'is', 'to',
'use', 'this', 'did', 'i', 'like', 'for']

-----Imperative:
18609
['fits', 'great', 'and', 'easy', 'to', 'clean', 'ca', 'complain', 'about', 'these', 'shoes', 'at', 'all', 'except',
'that', 'i', 'worried', 'about', 'getting', 'them']
```

13. Implement WordNetLemmatizer to lemmatize all tokens on both **Imperative** & **Question** sentences.
    - We here only care about verb, noun, adjective, adverb since these are most frequently appearing in the text.
    - Using WordNetLemmatizer to reduce the word forms to linguistically valid lemmas, in order that convert tokens into the form what user expected.
      e.g. cats -> cat, is -> be. etc.

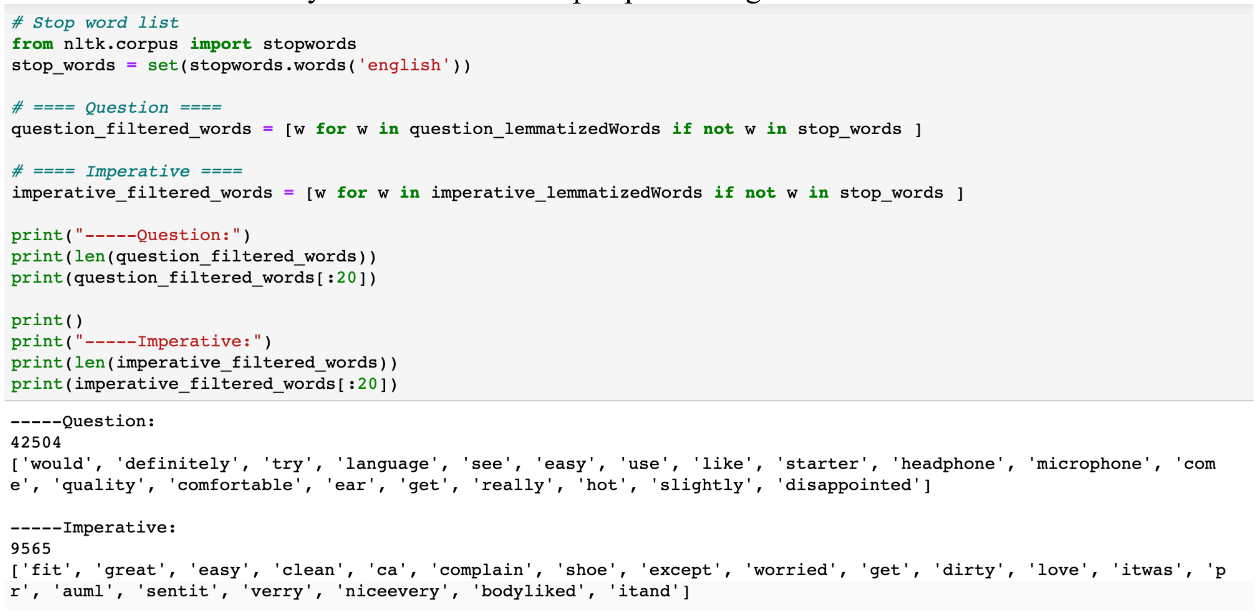
```python
# Lemmatization
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet

start = time.time()

def get_wordnet_pos(word):
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}

    return tag_dict.get(tag, wordnet.NOUN)

# Init Lemmatizer
lemmatizer = WordNetLemmatizer()

# ==== Question ====
question_lemmatizedWords = [lemmatizer.lemmatize(w, get_wordnet_pos(w)) for w in question_alphaWords]

# ==== Imperative ====
imperative_lemmatizedWords = [lemmatizer.lemmatize(w, get_wordnet_pos(w)) for w in imperative_alphaWords]
```



```
print('time:\t ' , time.time()-start)

print("-----Question:")
print(len(question_lemmatizedWords))
print(question_lemmatizedWords[:20])

print()
print("-----Imperative:")
print(len(imperative_lemmatizedWords))
print(imperative_lemmatizedWords[:20])
```

```
time:    20.58369517326355
-----Question:
91179
['i', 'would', 'definitely', 'try', 'other', 'language', 'after', 'see', 'just', 'how', 'easy', 'it', 'be', 'to', 'us
e', 'this', 'do', 'i', 'like', 'for']

-----Imperative:
18609
['fit', 'great', 'and', 'easy', 'to', 'clean', 'ca', 'complain', 'about', 'these', 'shoe', 'at', 'all', 'except', 'th
at', 'i', 'worried', 'about', 'get', 'them']
```

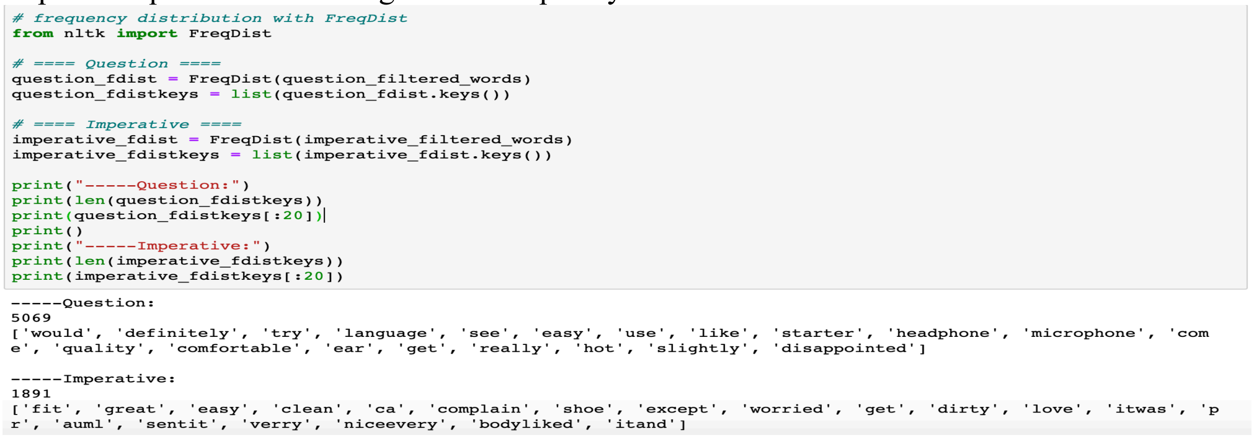
14. Import stopwords to remove all commonly used word such as “the”, “a”, “an”, “in”.
    - The reason using stopwords is we don’t want these words taking up space in our database because they are useless data as pre-processing.



```
# Stop word list
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))

# ==== Question ====
question_filtered_words = [w for w in question_lemmatizedWords if not w in stop_words ]

# ==== Imperative ====
imperative_filtered_words = [w for w in imperative_lemmatizedWords if not w in stop_words ]

print("-----Question:")
print(len(question_filtered_words))
print(question_filtered_words[:20])

print()
print("-----Imperative:")
print(len(imperative_filtered_words))
print(imperative_filtered_words[:20])
```

```
-----Question:
42504
['would', 'definitely', 'try', 'language', 'see', 'easy', 'use', 'like', 'starter', 'headphone', 'microphone', 'com
e', 'quality', 'comfortable', 'ear', 'get', 'really', 'hot', 'slightly', 'disappointed']

-----Imperative:
9565
['fit', 'great', 'easy', 'clean', 'ca', 'complain', 'shoe', 'except', 'worried', 'get', 'dirty', 'love', 'itwas', 'p
r', 'auml', 'sentit', 'verry', 'niceevery', 'bodyliked', 'itand']
```

15. Import FreqDist from nltk to generate frequency distribution.



```
# frequency distribution with FreqDist
from nltk import FreqDist

# ==== Question ====
question_fdist = FreqDist(question_filtered_words)
question_fdistkeys = list(question_fdist.keys())

# ==== Imperative ====
imperative_fdist = FreqDist(imperative_filtered_words)
imperative_fdistkeys = list(imperative_fdist.keys())

print("-----Question:")
print(len(question_fdistkeys))
print(question_fdistkeys[:20])
print()
print("-----Imperative:")
print(len(imperative_fdistkeys))
print(imperative_fdistkeys[:20])
```

```
-----Question:
5069
['would', 'definitely', 'try', 'language', 'see', 'easy', 'use', 'like', 'starter', 'headphone', 'microphone', 'com
e', 'quality', 'comfortable', 'ear', 'get', 'really', 'hot', 'slightly', 'disappointed']

-----Imperative:
1891
['fit', 'great', 'easy', 'clean', 'ca', 'complain', 'shoe', 'except', 'worried', 'get', 'dirty', 'love', 'itwas', 'p
r', 'auml', 'sentit', 'verry', 'niceevery', 'bodyliked', 'itand']
```

16. Based on Frequency Distribution, get most common tokens (Unigram). The result shows above.

```python
# 30 words by frequency

# ==== Question ====
print("-----Question:")
question_topkeys = question_fdist.most_common(30)
for pair in question_topkeys:
    print (pair)

# ==== Imperative ====
print()
print("-----Imperative:")
imperative_topkeys = imperative_fdist.most_common(30)
for pair in imperative_topkeys:
    print (pair)
```

17. Using Regular Expression to identify alphabet filter to retrieve alphabet tokens only.

```python
import re

def alpha_filter(w):
    pattern = re.compile('^[a-z]+$')
    if (pattern.match(w)):
        return False
    else:
        return True
```

18. Invoke collocations to generate bigrams among the reviews.

```python
from nltk.collocations import *
bigram_measures = nltk.collocations.BigramAssocMeasures()

# ==== Question ====
print("-----Question:")
question_finder = BigramCollocationFinder.from_words(question_lowercase)
question_scored = question_finder.score_ngrams(bigram_measures.raw_freq)

for bscore in question_scored[:20]:
    print(bscore)

# ==== Imperative ====
print()
print("-----Imperative:")
imperative_finder = BigramCollocationFinder.from_words(imperative_lowercase)
imperative_scored = imperative_finder.score_ngrams(bigram_measures.raw_freq)

for bscore in imperative_scored[:20]:
    print(bscore)
```

```
-----Question:
((',', 'i'), 0.00562429696287964)
((',', 'but'), 0.0044994375703037125)
(('?', 'i'), 0.004204831538914778)
(('?', '?'), 0.0034638527326941275)
(('?', ','), 0.003392433088721053)
((',', 'and'), 0.0031781741568018282)
(('&', '#'), 0.0031067545128287536)
(('#', '34'), 0.0029014230364061634)
(('34', ';'), 0.0029014230364061634)
(('it', "'s"), 0.0025443248165407896)
(('in', 'the'), 0.002508614994554252)
(('they', 'are'), 0.0023925580730980054)
(('of', 'the'), 0.0022586462406484903)
(('?', ')'), 0.0022140089631653187)
(('what', 'more'), 0.0021872265966754157)
(('do', "n't"), 0.001964040209259557)
(('i', "'m"), 0.0019194029317763852)
(('it', 'is'), 0.0018390558323066759)
(('?', 'the'), 0.0018301283768100416)
(('?', 'what'), 0.0018122734658167728)
```

```
-----Imperative:
(('!', ','), 0.01813663513101103)
(('!', '!'), 0.007760387147403758)
((',', 'i'), 0.007673191786196974)
(('!', 'looks'), 0.004577756463356149)
((',', 'and'), 0.0037929982124950954)
(('great', '!'), 0.0037058028512883113)
(('looks', 'great'), 0.0034006190870645684)
((',', 'this'), 0.0032698260452543927)
((',', 'but'), 0.0032262283646510006)
(('ca', "n't"), 0.002572263155600122)
(('they', 'are'), 0.002572263155600122)
((',', 'very'), 0.0024414701137899465)
(('in', 'the'), 0.0023978724331865544)
((',', 'the'), 0.0023542747525831624)
(('for', 'my'), 0.002179884030169595)
(('&', '#'), 0.0020490909883594194)
(('#', '34'), 0.0020054933077560274)
((',', 'these'), 0.0020054933077560274)
(('34', ';'), 0.0020054933077560274)
(('i', 'love'), 0.0018747002659458517)
```

19. Apply alpha_filter defined by Regular Expression previously.

```
# alpha bigrams

# ==== Question ====
print("-----Question:")
question_finder.apply_word_filter(alpha_filter)
question_scored1 = question_finder.score_ngrams(bigram_measures.raw_freq)
for bscore in question_scored1[:20]:
    print(bscore)

# ==== Imperative ====
print()
print("-----Imperative:")
imperative_finder.apply_word_filter(alpha_filter)
imperative_scored1 = imperative_finder.score_ngrams(bigram_measures.raw_freq)
for bscore in imperative_scored1[:20]:
    print(bscore)
```

```
-----Question:
(('in', 'the'), 0.002508614994554252)
(('they', 'are'), 0.0023925580730980054)
(('of', 'the'), 0.0022586462406484903)
(('what', 'more'), 0.0021872265966754157)
(('it', 'is'), 0.0018390558323066759)
(('i', 'have'), 0.0015980145338975486)
(('to', 'be'), 0.0015890870784009141)
(('a', 'little'), 0.0014998125234345708)
(('and', 'i'), 0.0014730301569446677)
(('for', 'a'), 0.001455175245951399)
(('is', 'a'), 0.0013748281464816899)
(('for', 'the'), 0.0013569732354884211)
(('and', 'the'), 0.0013480457799917868)
(('i', 'am'), 0.0013480457799917868)
(('on', 'the'), 0.0013480457799917868)
(('can', 'you'), 0.0013123359580052493)
(('i', 'say'), 0.0013123359580052493)
(('can', 'i'), 0.0012676986805220775)
(('if', 'you'), 0.0012319888585355402)
(('i', 'was'), 0.0011427143035691967)
```

```
-----Imperative:
(('looks', 'great'), 0.0034006190870645684)
(('they', 'are'), 0.002572263155600122)
(('in', 'the'), 0.0023978724331865544)
(('for', 'my'), 0.002179884030169595)
(('i', 'love'), 0.0018747002659458517)
(('it', 'is'), 0.0018747002659458517)
(('a', 'little'), 0.001743907224135676)
(('is', 'a'), 0.0017003095435322842)
(('and', 'i'), 0.0016131141823255003)
(('great', 'with'), 0.0015695165017221085)
(('i', 'have'), 0.0015695165017221085)
(('on', 'the'), 0.0015259188211187164)
(('this', 'is'), 0.0014823211405153246)
(('of', 'the'), 0.0014387234599119326)
(('to', 'wear'), 0.0014387234599119326)
(('to', 'be'), 0.0013951257793085408)
(('comfortable', 'and'), 0.001351528098705149)
(('for', 'a'), 0.001351528098705149)
(('i', 'was'), 0.001351528098705149)
(('fits', 'great'), 0.001307930418101757)
```

20. Remove stop words, to get Bigrams result, the result shows above.

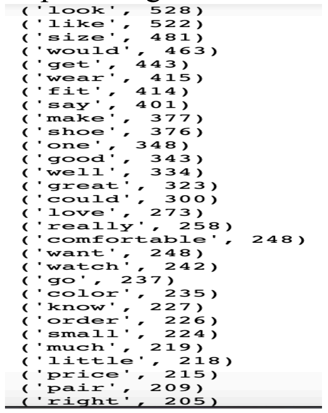
```
# stopword bigrams  -> Bigrams Result

# ==== Question ====
print("-----Question:")
question_finder.apply_word_filter(lambda w: w in stop_words)
question_scored2 = question_finder.score_ngrams(bigram_measures.raw_freq)
for bscore in question_scored2[:20]:
    print(bscore)

# ==== Imperative ====
print()
print("-----Imperative:")
imperative_finder.apply_word_filter(lambda w: w in stop_words)
imperative_scored2 = imperative_finder.score_ngrams(bigram_measures.raw_freq)
for bscore in imperative_scored2[:20]:
    print(bscore)
```

Cleaning:

We need to clean all stopwords on each sentence.
In question sentence, we need to remove exclamation marks.
In imperative sentence, we need to remove question marks.

Interpretation:

- **Question Sentence:**
  1. Based on the table result, there're 3682 question sentences. Average length tokens in each is around 30. Percentage in total is around 0.4%. By the statistics above, we could analyze that customers usually type about 30 vocabularies to ask a question, meaning that they do care about the product. We could image that if a customer asks a question within a few tokens, they might not really care about the products, but they just ask about it. The number of question sentences exists in total is not that much, but it still seems that there're a lot of customers ask questions, we could assume that comparing other types of sentences, question sentences could be one of the largest scales in the total amount of all sentences. In conclusion, customers probably more care about all issues of products, and they ask questions on them.

  2. Unigram Frequency Analysis:
     Top 30 unigrams show following:


```
('look', 528)
('like', 522)
('size', 481)
('would', 463)
('get', 443)
('wear', 415)
('fit', 414)
('say', 401)
('make', 377)
('shoe', 376)
('one', 348)
('good', 343)
('well', 334)
('great', 323)
('could', 300)
('love', 273)
('really', 258)
('comfortable', 248)
('want', 248)
('watch', 242)
('go', 237)
('color', 235)
('know', 227)
('order', 226)
('small', 224)
('much', 219)
('little', 218)
('price', 215)
('pair', 209)
('right', 205)
```

     Based on the result above, it obviously shows that most of them are verb, and most of them relate to the products such as "look, wear, fit, love". And some of them are adjective/adverb such as well, good, great, comfortable. Based on the data, we could know that customers ask questions positively and they might be interested in the products than we expected.

  3. Bigram Frequency Analysis:
     Top 20 unigrams show following:

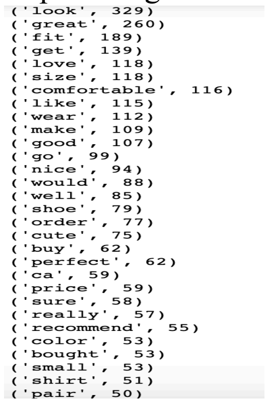
```
(('go', 'wrong'), 0.00031246094238220225)
(('well', 'made'), 0.00028567857589229916)
(('year', 'old'), 0.00024104129840912743)
(('look', 'like'), 0.00023211384291249308)
(('good', 'quality'), 0.0001964040209259557)
(('looks', 'great'), 0.0001964040209259557)
(('would', 'recommend'), 0.00018747656542932134)
(('feel', 'like'), 0.000178549109932687)
(('another', 'pair'), 0.00016962165443605264)
(('high', 'quality'), 0.0001606941989394183)
(('make', 'sure'), 0.0001606941989394183)
(('highly', 'recommend'), 0.00015176674344278393)
(('fits', 'great'), 0.00014283928794614958)
(('little', 'bit'), 0.00014283928794614958)
(('looks', 'like'), 0.00014283928794614958)
(('much', 'better'), 0.00014283928794614958)
(('size', 'smaller'), 0.00013391183244951523)
(('years', 'ago'), 0.00013391183244951523)
(('look', 'good'), 0.00012498437695288088)
(('one', 'size'), 0.00012498437695288088)
```

Based on the result, we could see the most bigram is negative which is "go wrong", whereas, most of others are still positive bigrams. I cannot make prediction why "go wrong" is the most bigram, but I could guess that it could be indicated to delivery or something happened wrong. However, based on other positive bigrams, we still could obviously see that customer ask questions positively and they might be satisfied with products or interested in the products so much.

- **Imperative Sentences**

1. Based on the table result, there're only 1127 imperative sentences in the total. The average length by tokens is about 20, and the percentage in the total is 0.12%. Comparing question sentences, it's extremely less than them. Customers probably don't have much requests to sellers, the reason I guess is that they don't expect sellers could do anything or change minds on their requests, but they could ask sellers questions to ask more details on products and make decisions then. In imperative sentences, customer only type around 20 vocabularies averagely. It's less than question sentences. We still could speculate that customers do not care about requests again.

2. Unigram Frequency Analysis:
   Top 30 unigrams show following:



```
('look', 329)
('great', 260)
('fit', 189)
('get', 139)
('love', 118)
('size', 118)
('comfortable', 116)
('like', 115)
('wear', 112)
('make', 109)
('good', 107)
('go', 99)
('nice', 94)
('would', 88)
('well', 85)
('shoe', 79)
('order', 77)
('cute', 75)
('buy', 62)
('perfect', 62)
('ca', 59)
('price', 59)
('sure', 58)
('really', 57)
('recommend', 55)
('color', 53)
('bought', 53)
('small', 53)
('shirt', 51)
('pair', 50)
```

   Based on the result, we could see that most of them are positive unigrams. Customers are satisfied with the products by make requests (as imperative). The words such that "great, love, comfortable, good, nice, etc." could show how customers like them. Also, we could analyze that customers more care about quality and outlook of products.

3. Bigram Frequency Analysis:
   Top 20 unigrams show following:
```
(('looks', 'great'), 0.0034006190870645684)
(('fits', 'great'), 0.001307930418101757)
(('go', 'wrong'), 0.001264332737498365)
(('make', 'sure'), 0.0010463443344814055)
(('looks', 'good'), 0.0009155512926712299)
(('look', 'great'), 0.0006975628896542704)
(('would', 'recommend'), 0.0006103675284474866)
(('fits', 'perfect'), 0.000435976806033919)
(('well', 'made'), 0.000435976806033919)
(('would', 'buy'), 0.000435976806033919)
(('fit', 'great'), 0.0003923791254305271)
(('good', 'quality'), 0.0003923791254305271)
(('highly', 'recommend'), 0.0003923791254305271)
(('super', 'cute'), 0.0003923791254305271)
(('feels', 'great'), 0.0003487814448271352)
(('great', 'price'), 0.0003487814448271352)
(('say', 'enough'), 0.0003487814448271352)
(('fits', 'well'), 0.0003051837642237433)
(('look', 'good'), 0.0003051837642237433)
(('look', 'like'), 0.0003051837642237433)
```
   Based on the result, we could see that most unigrams are positive, and it also still shows that customers are satisfied with the products. Unlike question sentence, the most bigram is negative, but in imperative sentences, it looks that the negative bigram is the third most bigram, so we could expect that customers are still positive to the products except some of them do not satisfy with it. Meanwhile, apparently customer smore care about the quality, outlook, and how the products fit on their own.

- **Summary:**
  Based on the interpretation on both question and imperative sentences, I learned that customers are positively satisfied with the products and they do more care about the quality & outlook of products.

Envision how to conduct a sentiment analysis on review texts

Adjective phrase is important and necessary to generate sentiment from sentences. For sentiment, we could see how customers' expectations regarding on products they have reviewed. Without sentiment, it's hard to predict what customers thought and care about. Think about that the non-sentiment imperative sentence, e.g. "Do you know?", what can we get from it? So to conduct sentiment sentences, we should primally extract sentences with adjective phrases.

Another case is that we need to stop unnecessary Stopwords from reviews to analyze whether the reviews are sentiment or not. **Stopwords** are just simple words that don't have any meaningful value to be analyzed, so we could simplify remove them directly.

To achieve the goals to build sentiment, we also need to care about how customers feels and thoughts, such that whether they're satisfied, whether they have any complaints, or they feel nothing of products. To do this, I think the better way is that we could extract **Positive/Negative/Natural** sets, and then analyze each sentence with sets. We could get more accurate sentiment if we do this way.

In conclusion, we need to include adjective phrases, remove Stopwords, and add sets of positive/negative/natural.