In [144]:
```
pwd
```
Out[144]:
```
'C:\\Users\\Dell\\downloads\\character-extraction-master\\character-extraction-master'
```
In [4]:
```
cd character-extraction-master
```
```
C:\Users\Dell\downloads\character-extraction-master\character-extraction-master
```
In [5]:
```
import json
import nltk
import re

from collections import defaultdict
from nltk.corpus import stopwords
from pattern.en import parse, Sentence, mood
from pattern.db import csv
from pattern.vector import Document, NB
```
In [111]:
```
import re
text=text.replace("Harry Potter and the Philosophers Stone - J.K."," ")
text=text.replace("Page |"," ")
text=text.replace("Page"," ")
text=text.replace("Rowling"," ")
text = re.sub(r'[0-9]+', '', text)
text=re.sub('\s+',' ',text)

with open("hp_ps_co2.txt", "w",encoding="utf-8") as f:
    f.write(text)
    f.close()
```
In [150]:
```
def readText():
    """
    Reads the text from a text file.
    """
    with open("hp_ps_co2.txt", "rb") as f:
        text = f.read().decode('utf-8-sig')
    return text
```

In [7]:
```
def chunkSentences(text):
    """
    Parses text into parts of speech tagged with parts of speech labels.

    Used for reference: https://gist.github.com/onyxfish/322906
    """
    sentences = nltk.sent_tokenize(text)
    tokenizedSentences = [nltk.word_tokenize(sentence)
                          for sentence in sentences]
    taggedSentences = [nltk.pos_tag(sentence)
                       for sentence in tokenizedSentences]
    if nltk.__version__[0:2] == "2.":
        chunkedSentences = nltk.batch_ne_chunk(taggedSentences, binary=True)
    else:
        chunkedSentences = nltk.ne_chunk_sents(taggedSentences, binary=True)
    return chunkedSentences
```

In [8]:
```
nltk.download('averaged_perceptron_tagger')
```

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

Out[8]:
```
True
```

In [9]:
```python
def extractEntityNames(tree, _entityNames=None):
    """
    Creates a local list to hold nodes of tree passed through, extracting named
    entities from the chunked sentences.

    Used for reference: https://gist.github.com/onyxfish/322906
    """
    if _entityNames is None:
        _entityNames = []
    try:
        if nltk.__version__[0:2] == "2.":
            label = tree.node
        else:
            label = tree.label()
    except AttributeError:
        pass
    else:
        if label == 'NE':
            _entityNames.append(' '.join([child[0] for child in tree]))
        else:
            for child in tree:
                extractEntityNames(child, _entityNames=_entityNames)
    return _entityNames
```

In [10]:
```python
def buildDict(chunkedSentences, _entityNames=None):
    """
    Uses the global entity list, creating a new dictionary with the properties
    extended by the local list, without overwriting.

    Used for reference: https://gist.github.com/onyxfish/322906
    """
    if _entityNames is None:
        _entityNames = []

    for tree in chunkedSentences:
        extractEntityNames(tree, _entityNames=_entityNames)

    return _entityNames
```

In [11]:
```
nltk.download('maxent_ne_chunker')
```

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

Out[11]:
```
True
```

In [12]:
```
def removeStopwords(entityNames, customStopWords=None):
    """
    Brings in stopwords and custom stopwords to filter mismatches out.
    """
    # Memoize custom stop words
    if customStopWords is None:
        with open("customStopWords.txt", "rb") as f:
            customStopwords = f.read().split(', ')

    for name in entityNames:
        if name in stopwords.words('english') or name in customStopwords:
            entityNames.remove(name)
```

In [13]:
```
nltk.download('words')
```

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

Out[13]:
```
True
```

In [14]:
```
def getMajorCharacters(entityNames):
    """
    Adds names to the major character list if they appear frequently.
    """
    return {name for name in entityNames if entityNames.count(name) > 15}
```

In [15]:
```python
def splitIntoSentences(text):
    """
    Split sentences on .?! "" and not on abbreviations of titles.
    Used for reference: http://stackoverflow.com/a/8466725
    """
    sentenceEnders = re.compile(r"""
    # Split sentences on whitespace between them.
    (?:               # Group for two positive lookbehinds.
      (?<=[.!?])      # Either an end of sentence punct,
    | (?<=[.!?]['"])  # or end of sentence punct and quote.
    )                 # End group of two positive lookbehinds.
    (?<!  Mr\.   )    # Don't end sentence on "Mr."
    (?<!  Mrs\.  )    # Don't end sentence on "Mrs."
    (?<!  Ms\.   )    # Don't end sentence on "Ms."
    (?<!  Jr\.   )    # Don't end sentence on "Jr."
    (?<!  Dr\.   )    # Don't end sentence on "Dr."
    (?<!  Prof\. )    # Don't end sentence on "Prof."
    (?<!  Sr\.   )    # Don't end sentence on "Sr."
    \s+               # Split on whitespace between sentences.
    """, re.IGNORECASE | re.VERBOSE)
    return sentenceEnders.split(text)
```

In [16]:
```python
def compareLists(sentenceList, majorCharacters):
    """
    Compares the list of sentences with the character names and returns
    sentences that include names.
    """
    characterSentences = defaultdict(list)
    for sentence in sentenceList:
        for name in majorCharacters:
            if re.search(r"\b(?=\w)%s\b(?!\w)" % re.escape(name),
                         sentence,
                         re.IGNORECASE):
                characterSentences[name].append(sentence)
    return characterSentences
```

In [17]:
```
def extractMood(characterSentences):
    """
    Analyzes the sentence using grammatical mood module from pattern.
    """
    characterMoods = defaultdict(list)
    for key, value in characterSentences.items():
        for x in value:
            characterMoods[key].append(mood(Sentence(parse(str(x),
                                                        lemmata=True))))
    return characterMoods
```

In [18]:
```
def extractSentiment(characterSentences):
    """
    Trains a Naive Bayes classifier object with the reviews.csv file, analyzes
    the sentence, and returns the tone.
    """
    nb = NB()
    characterTones = defaultdict(list)
    for review, rating in csv("reviews.csv"):
        nb.train(Document(review, type=int(rating), stopwords=True))
    for key, value in characterSentences.items():
        for x in value:
            characterTones[key].append(nb.classify(str(x)))
    return characterTones
```

In [19]:
```
def writeAnalysis(sentenceAnalysis):
    """
    Writes the sentence analysis to a text file in the same directory.
    """
    with open("sentenceAnalysis.txt", "wb") as f:
        for item in sentenceAnalysis.items():
            f.write("%s:%s\n" % item)
```

In [20]:
```
def writeToJSON(sentenceAnalysis):
    """
    Writes the sentence analysis to a JSON file in the same directory.
    """
    with open("sentenceAnalysis.json", "wb") as f:
        json.dump(sentenceAnalysis, f)
```

```
In [21]: import nltk
         nltk.download('stopwords')
         from nltk.corpus import stopwords

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

```
In [22]: !pip install many_stop_words

    Requirement already satisfied: many_stop_words in c:\users\dell\anaconda3\lib\site-packages (0.2.2)

    WARNING: You are using pip version 19.1.1, however version 20.0.2 is available.
    You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

```
In [37]: from many_stop_words import get_stop_words
         stop_list=list(get_stop_words('en'))        #About 900 stop words
         nltk_words = list(stopwords.words('english'))   #About 150 stop words
         stop_list.extend(nltk_words)
         stop_list += ["Yeh"]
```

```
In [151]: text = readText()
```

```
In [152]: chunkedSentences = chunkSentences(text)
```

```
In [153]: entityNames = buildDict(chunkedSentences)
```

```
In [154]: entNames = list(set(entityNames))
          len(entNames)

Out[154]: 810
```

```
In [155]:  majorCharacters = getMajorCharacters(entityNames)
          len(majorCharacters)
          majCharacters=list(majorCharacters)
```

```
In [156]: majorCharacters
```

```
Out[156]: {'Albus Dumbledore',
           'Aunt Petunia',
           'Bane',
           'Crabbe',
           'Dudley',
           'Dumbledore',
           'Dursleys',
           'Fang',
           'Fat Lady',
           'Filch',
           'Fluffy',
           'Fred',
           'George',
           'George Weasley',
           'Goyle',
           'Great Hall',
           'Gringotts',
           'Gryffindor',
           'Hagrid',
           'Harry',
           'Harry Harry',
           'Harry Potter',
           'Hedwig',
           'Hermione',
           'Hermione Granger',
           'Hogwarts',
           'House',
           'Lord Voldemort',
           'Madam Hooch',
           'Madam Pomfrey',
           'Malfoy',
           'Mirror',
           'Mr. Dursley',
           'Mr. Ollivander',
           'Muggle',
           'Neville',
           'Nicolas Flamel',
           'Norbert',
           'Oliver Wood',
           'Peeves',
           'Percy',
           'Philosophers Stone',
           'Potter',
           'Privet Drive',
           'Professor',
           'Professor McGonagall',
           'Professor Snape',
           'Quaffle',
           'Quidditch',
           'Quirrell',
           'Ron',
           'Ronan',
           'Seamus',
           'Slytherin',
           'Snape',
           'Sorcerer',
           'Uncle Vernon',
           'Voldemort',
           'Weasley',
           'Wood',
           'Yeh'}
```

```
In [160]: len(majorCharacters)
```

```
Out[160]: 60
```

```
In [159]: for name in majorCharacters:
                  if name in stop_list:
                      majorCharacters.remove(name)
```

```
In [161]: sentenceList = splitIntoSentences(text)
```

```
In [162]: characterSentences = compareLists(sentenceList, majorCharacters)
```

```
In [137]: characterMoods = defaultdict(list)
          for key, value in characterSentences.items():
                  for x in value:
                      continue
                      #characterMoods[key].append(mood(Sentence(parse(str(x)))))
```

```
In [163]: sentenceAnalysis = defaultdict(list,
                                     [(k,characterSentences[k])
                                      for k in characterSentences])
```

In [164]:
```
with open("sentenceAnalysis_hp_co.txt", "w",encoding="utf-8") as f:
    for k in characterSentences:
        f.write("\n\n----------------------------------------------\n\n")
        f.write("\n*************************************************\n")
        f.write(k)
        f.write("\n*************************************************\n")
        f.write("\n----------------------------------------------\n")
        for x in characterSentences[k]:
            f.write(x)
            f.write("\n----------------------------------------------\n")
```

In [ ]:
```
#with open("sentenceAnalysis_hp.txt", "w") as f:
      # for item in sentenceAnalysis.items():
            #f.write("%s:%s\n" % item)
```

In [165]:
```
with open("sentenceAnalysis.json", "w") as f:
        json.dump(sentenceAnalysis, f)
```

In [166]:
```
import spacy
import en_core_web_sm
nlp = en_core_web_sm.load()
```

In [167]:
```
with open("charecter_freq.txt", "w",encoding="utf-8") as f:
    for name in majorCharacters:
        f.write(name)
        f.write(":\n")
        f.write(str(entityNames.count(name)))
        f.write("\n--------------------------------------------------\n")
```

In [168]:
```
lines_list = open('pa.txt').read().splitlines()
```

In [169]:
```
type(lines_list[0])
```

Out[169]:
```
str
```

In [170]:
```
characterDescription = defaultdict(set)
```

In [171]:
```python
def compareLists1(sentenceList1, majorCharacters1,k):
    """
    Compares the list of sentences with the character names and returns
    sentences that include names.
    """
    for sentence in sentenceList1:
        for name in majorCharacters1:
            if re.search(name,sentence,re.IGNORECASE):
                characterDescription[k].add(sentence)
```

In [172]:
```python
for k in characterSentences:
    compareLists1(characterSentences[k],lines_list,k)
```

In [175]:
```python
z
```

Out[175]:
```
3602
```

In [174]:
```python
for k in characterDescription:
    print("-----------------------------------------------")
    print("*********************************************")
    print(k)
    print("*********************************************")
    print("-----------------------------------------------")
    for x in characterDescription[k]:
        print(x)
        z=z+1
        print("-----------------------------------------------")
```

```
the cat on the wall outside was sitting as still as a statue, the cat on the wall outside eyes fixed unblinkingly on the far c
orner of Privet Drive.
----------------------------------------------
Albus Dumbledore clicked the silver Put- Outer once, and twelve balls of light sped back to their street lamps so that Privet
Drive glowed suddenly orange and Albus Dumbledore could make out a tabby cat slinking around the corner at the other end of th
e street.
----------------------------------------------
Nothing like A man had ever been seen on Privet Drive.
----------------------------------------------
Mr. Dursley gave Mr. Dursley a little shake and put a tabby cat standing on the corner of Privet Drive out of Mr. Dursley min
d.
----------------------------------------------
Harry Potter couldn’t know that at this very moment, people meeting in secret all over the country were holding up people glas
ses and saying in hushed voices: “To Harry Potter – the boy who lived!” THE VANASHIG GLASS Nearly ten years had passed since t
he Dursleys had woken up to find their nephew on the front step, but Privet Drive had hardly changed at all.
----------------------------------------------
Potter, The Smallest Bedroom, Privet Drive – ’ ” With a strangled cry, Uncle Vernon leapt from Uncle Vernon seat and ran down
the hall, Harry right behind Harry.
----------------------------------------------
Potter The Cupboard under the Stairs Privet Drive Little Whinging Surrey The envelope was thick and heavy, made of yellowish p
```

In [178]:
```python
with open("description.txt", "w",encoding="utf-8") as f:
    for k in characterDescription:
        f.write("\n-----------------------------------------------\n")
        f.write("\n*********************************************\n")
        f.write(k)
        f.write("\n*********************************************\n")
        f.write("\n-----------------------------------------------\n")
        for x in characterDescription[k]:
            f.write(x)
            f.write("\n-----------------------------------------------\n")
```

```
In [179]: def apply_extraction(row):
              doc=nlp(row)


              ## FIRST RULE OF DEPENDANCY PARSE -
              ## M - Sentiment modifier || A - Aspect
              ## RULE = M is child of A with a relationshio of amod
              rule1_pairs = []
              for token in doc:
                  if token.dep_ == "amod":
                      rule1_pairs.append((token.head.text, token.text))
                      #return row['height'] * row['width']


              ## SECOND RULE OF DEPENDANCY PARSE -
              ## M - Sentiment modifier || A - Aspect
              #Direct Object - A is a child of something with relationship of nsubj, while
              # M is a child of the same something with relationship of dobj
              #Assumption - A verb will have only one NSUBJ and DOBJ

              rule2_pairs = []
              for token in doc:
                  children = token.children
                  A = "999999"
                  M = "999999"
                  for child in children :
                      if(child.dep_ == "nsubj"):
                          A = child.text
                      if(child.dep_ == "dobj"):
                          M = child.text
                  if(A != "999999" and M != "999999"):
                      rule2_pairs.append((A, M))


              ## THIRD RULE OF DEPENDANCY PARSE -
              ## M - Sentiment modifier || A - Aspect
              #Adjectival Complement - A is a child of something with relationship of nsubj, while
              # M is a child of the same something with relationship of acomp
              #Assumption - A verb will have only one NSUBJ and DOBJ

              rule3_pairs = []

              for token in doc:

                  children = token.children
```

```python
        A = "999999"
        M = "999999"
        for child in children :
            if(child.dep_ == "nsubj"):
                A = child.text

            if(child.dep_ == "acomp"):
                M = child.text

        if(A != "999999" and M != "999999"):
            rule3_pairs.append((A, M))

    ## FOURTH RULE OF DEPENDANCY PARSE -
    ## M - Sentiment modifier || A - Aspect

    #Adverbial modifier to a passive verb - A is a child of something with relationship of nsubjpass, while
    # M is a child of the same something with relationship of advmod

    #Assumption - A verb will have only one NSUBJ and DOBJ

    rule4_pairs = []
    for token in doc:


        children = token.children
        A = "999999"
        M = "999999"
        for child in children :
            if(child.dep_ == "nsubjpass"):
                A = child.text

            if(child.dep_ == "advmod"):
                M = child.text

        if(A != "999999" and M != "999999"):
            rule4_pairs.append((A, M))


    ## FIFTH RULE OF DEPENDANCY PARSE -
    ## M - Sentiment modifier || A - Aspect

    #Complement of a copular verb - A is a child of M with relationship of nsubj, while
    # M has a child with relationship of cop

    #Assumption - A verb will have only one NSUBJ and DOBJ
    rule5_pairs = []
    for token in doc:
        children = token.children
        A = "999999"
        buf_var = "999999"
        for child in children :
            if(child.dep_ == "nsubj"):
                A = child.text

            if(child.dep_ == "cop"):
                buf_var = child.text

        if(A != "999999" and buf_var != "999999"):
            rule3_pairs.append((A, token.text))

    aspects = []
    aspects = rule1_pairs + rule2_pairs + rule3_pairs +rule4_pairs +rule5_pairs
    dic = {"aspect_pairs" : aspects}
    return dic
```

In [180]:
```
noun_adj_pairs = defaultdict(list)
```

```
In [181]: for k in characterDescription:
              print("-----------------------------------------------")
              print("***********************************************")
              print(k)
              print("***********************************************")
              print("-----------------------------------------------")
              for x in characterDescription[k]:
                  print(apply_extraction(x))
                  print("-----------------------------------------------")
```

```
-----------------------------------------------
{'aspect_pairs': [('place', 'first')]}
-----------------------------------------------
{'aspect_pairs': [('Voldemort', 'mistakes')]}
-----------------------------------------------
{'aspect_pairs': []}
-----------------------------------------------
{'aspect_pairs': []}
-----------------------------------------------
{'aspect_pairs': [('I', 'right')]}
-----------------------------------------------
{'aspect_pairs': [('Voldemort', 'me')]}
-----------------------------------------------

{'aspect_pairs': [('Quirrell', 'full'), ('Quirrell', 'you')]}
-----------------------------------------------
{'aspect_pairs': [('Voldemort', 'mother')]}
-----------------------------------------------
{'aspect_pairs': [('second', 'next'), ('Voldemort', 'HIM')]}
-----------------------------------------------
```

```
In [183]: for k in characterDescription:
              for x in characterDescription[k]:
                  noun_adj_pairs[k].append(apply_extraction(x))
```

In [184]:
```
noun_adj_pairs
```

```
                  ('Potters', 'son')]},
                {'aspect_pairs': [('thing', 'first'),
                  ('cat', 'tabby'),
                  ('it', 'mood')]},
                {'aspect_pairs': [('tin', 'single'),
                  ('tin', 'collecting'),
                  ('Dursley', 'tin')]},
                {'aspect_pairs': [('people', 'different'),
                  ('doughnut', 'large'),
                  ('words', 'few'),
                  ('people', 'different'),
                  ('Dursley', 'words'),
                  ('people', 'what')]},
                {'aspect_pairs': [('cat', 'tabby'),
                  ('one', 'same'),
                  ('cat', 'tabby'),
                  ('markings', 'same'),
                  ('cat', 'tabby'),
                  ('cat', 'markings'),
                  ('he', 'eyes'),
```

In [189]:
```python
with open("description_noun_adj.txt", "w",encoding="utf-8") as f:
    f.write(str(noun_adj_pairs))
```

In [188]:
```python
import json

json = json.dumps(noun_adj_pairs)
f = open("description_noun_adj.json","w")
f.write(json)
f.close()
```

In [ ]: