```
import string
import nltk
from nltk.tokenize import sent_tokenize, word_tokenize
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
from nltk import RegexpParser
from nltk.corpus import wordnet as wn
from nltk.wsd import lesk
import codecs
from nltk.chunk import conlltags2tree, tree2conlltags
from pprint import pprint
import spacy
from spacy import displacy
from collections import Counter
import en_core_web_sm
import os
import re
from nltk.sem.relextract import extract_rels, rtuple
import seaborn as sn
import pandas as pd
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import ieer


class doc():
```

```
  pass



def apostropheToWords():
    contractions = {
        "ain't": "am not",
        "aren't": "are not",
        "can't": "cannot",
        "can't've": "cannot have",
        "'cause": "because",
        "could've": "could have",
        "couldn't": "could not",
        "couldn't've": "could not have",
        "didn't": "did not",
        "doesn't": "does not",
        "don't": "do not",
        "hadn't": "had not",
        "hadn't've": "had not have",
        "hasn't": "has not",
        "haven't": "have not",
        "he'd": "he had",
        "he'd've": "he would have",
        "he'll": "he shall",
        "he'll've": "he shall have",
```

"he's": "he is",

"how'd": "how did",

"how'd'y": "how do you",

"how'll": "how will",

"how's": "how has",

"i'd": "I had",

"i'd've": "I would have",

"i'll": "I shall",

"i'll've": "I shall have",

"i'm": "I am",

"i've": "I have",

"isn't": "is not",

"it'd": "it had",

"it'd've": "it would have",

"it'll": "it shall",

"it'll've": "it shall have",

"it's": "it has",

"let's": "let us",

"ma'am": "madam",

"mayn't": "may not",

"might've": "might have",

"mightn't": "might not",

"mightn't've": "might not have",

"must've": "must have",

"mustn't": "must not",

"mustn't've": "must not have",

"needn't": "need not",

"needn't've": "need not have",

"o'clock": "of the clock",

"oughtn't": "ought not",

"oughtn't've": "ought not have",

"shan't": "shall not",

"sha'n't": "shall not",

"shan't've": "shall not have",

"she'd": "she had",

"she'd've": "she would have",

"she'll": "she shall",

"she'll've": "she shall have",

"she's": "she has",

"should've": "should have",

"shouldn't": "should not",

"shouldn't've": "should not have",

"so've": "so have",

"so's": "so as",

"that'd": "that would",

"that'd've": "that would have",

"that's": "that has",

"there'd": "there had",

"there'd've": "there would have",

"there's": "there has",

"they'd": "they had",

"they'd've": "they would have",

"they'll": "they shall",

"they'll've": "they shall have",

"they're": "they are",

"they've": "they have",

"to've": "to have",

"wasn't": "was not",

"we'd": "we had",

"we'd've": "we would have",

"we'll": "we will",

"we'll've": "we will have",

"we're": "we are",

"we've": "we have",

"weren't": "were not",

"what'll": "what shall",

"what'll've": "what shall have",

"what're": "what are",

"what's": "what has",

"what've": "what have",

"when's": "when has",

"when've": "when have",

"where'd": "where did",

"where's": "where has",

"where've": "where have",

```
    "who'll": "who shall",
    "who'll've": "who will have",
    "who's": "who has",
    "who've": "who have",
    "why's": "why has",
    "why've": "why have",
    "will've": "will have",
    "won't": "will not",
    "won't've": "will not have",
    "would've": "would have",
    "wouldn't": "would not",
    "wouldn't've": "would not have",
    "y'all": "you all",
    "y'all'd": "you all would",
    "y'all'd've": "you all would have",
    "y'all're": "you all are",
    "y'all've": "you all have",
    "you'd": "you had",
    "you'd've": "you would have",
    "you'll": "you will",
    "you'll've": "you will have",
    "you're": "you are",
    "you've": "you have"
}
```

```
    return contractions


def preprocessBook1():

    doc = codecs.open('1661-0.txt','r','utf-8')

    start = '\nA SCANDAL IN BOHEMIA\r\n\r\n\r\nI.\n'

    end = "\nEnd of the Project Gutenberg EBook of The Adventures of Sherlock Holmes, by \r\nArthur Conan Doyle\r\n\r\n*** END OF THIS PROJECT GUTENBERG EBOOK THE ADVENTURES OF SHERLOCK HOLMES ***\r\n\r\n***** This file should be named 1661-0.txt or 1661-0.zip *****\r\nThis and all associated files of various formats will be found in:\r\n http://www.gutenberg.org/1/6/6/1661/\r\n\r\nProduced by an anonymous Project Gutenberg volunteer and Jose Menendez\r\n\r\nUpdated editions will replace the previous one--the old editions will\r\nbe renamed.\n"


    sentences = getSentences(doc,start,end,False)


    return sentences


def preprocessBook2():

    doc = codecs.open('120-0.txt','r','utf-8')

    start = '\nAND LAST .\n'

    end = "\nPieces of eight!”\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\nEnd of Project Gutenberg's Treasure Island, by Robert Louis Stevenson\r\n\r\n*** END OF THIS PROJECT GUTENBERG EBOOK TREASURE ISLAND ***\r\n\r\n***** This file should be named 120-0.txt or 120-0.zip *****\r\nThis and all associated files of various formats will be found in:\r\n        http://www.gutenberg.org/1/2/120/\r\n\r\nProduced by Judy Boss, John Hamm and David Widger\r\n\r\nUpdated editions will replace the previous one--the old editions\r\nwill be renamed.\n"

    sentences = getSentences(doc,start,end,False)


    #print(sentences)
```

```
    return sentences


def tokenize(sentences):
    tokenizer = RegexpTokenizer(r'\w+')
    tokens = {}
    tokenized_sentences = []
    for sentence in sentences:
        words = tokenizer.tokenize(sentence)
        for word in words:
            if word.lower() not in tokens.keys():
                tokens[word.lower()] = 1
            else:
                tokens[word.lower()] += 1


        tokenized_sentences.append(words)


    return tokens,tokenized_sentences


def makeWordCloud(tokens):
    wordcloud = WordCloud(width = 800, height = 800,
                background_color = 'black',
                min_font_size = 2).fit_words(tokens)


    plt.figure(figsize = (8, 8), facecolor = None)
    plt.imshow(wordcloud)
```

```
    plt.axis("off")
    plt.tight_layout(pad = 0)

    plt.show()

def removeStopWords(tokens):
    stop_words = set(stopwords.words('english'))
    tokens1 = tokens.copy()
    for word in stop_words:
        if word in tokens1.keys():
            del tokens1[word]

    makeWordCloud(tokens1)

    return tokens1

def plot(X,Y,xlabel,ylabel,title):
    plt.bar(X, Y, tick_label = X, width = 0.8, color = ['red', 'green'])
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    plt.show()

def plotRelationShip(tokens):
    word_lengths = {}
```

```
    for i in tokens.keys():
        if len(i) not in word_lengths.keys():
            word_lengths[len(i)] = tokens[i]
        else:
            word_lengths[len(i)] += tokens[i]

    X = []
    Y = []

    for i in word_lengths.keys():
        X.append(i)

    X.sort()

    for i in X:
        Y.append(word_lengths[i])

    xlabel = 'word length'
    ylabel = 'frequency'
    title = 'Relationship between word length and frequency'
    plot(X,Y,xlabel,ylabel,title)


def PoSTagging(sentences):
    tags = {}
```

```
    for wordList in sentences:
        word_tags = nltk.pos_tag(wordList)
        for i in word_tags:
            if i[1] not in tags.keys():
                tags[i[1]] = 1
            else:
                tags[i[1]] += 1

    X = []
    Y = []
    for i in tags.keys():
        X.append(i)

    for i in X:
        Y.append(tags[i])

    xlabel = 'tags'
    ylabel = 'frequency'
    title = 'Relationship between tags and frequency'

    plot(X,Y,xlabel,ylabel,title)

def plotTop20Words(tokens):
    words = []
    for i in tokens.keys():
```

```
        words.append((tokens[i],i))


    words.sort(reverse = True)
    X = []
    Y = []
    for i in words:
        X.append(i[1])
        Y.append(i[0])
        if len(X) == 20:
            break


    xlabel = 'words'
    ylabel = 'frequency'
    title = 'Top 20 words in the book'
    plot(X,Y,xlabel,ylabel,title)


def getSentences(doc,start,end,test):
    tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
    content = doc.read()
    text = '\n-----\n'.join(tokenizer.tokenize(content))
    sentences = text.split('-----')
    #print(sentences)
    if not test:
        while True:
            if sentences[0] == start:
```

```
            sentences.pop(0)
            break
        sentences.pop(0)

    while True:
        if sentences[-1] == end:
            sentences.pop()
            break
        sentences.pop()

  pure_sentences = []
  for sentence in sentences:
    pure_sentences.append(sentence.replace('\n', ' ').replace('\r', '').replace('“',
'').replace('”','').replace('—',''))

  return pure_sentences

def findCategories(tokens,tags,nouns,verbs):
  for word in tokens:
    if not lesk(tokens,word):
      continue
    if lesk(tokens, word).pos() == 'n':
      category = lesk(tokens, word).lexname()
      if category not in nouns.keys():
        nouns[category] = 1
      else:
```

```
            nouns[category] += 1
        elif lesk(tokens, word).pos() == 'v':
            category = lesk(tokens, word).lexname()
            if category not in verbs.keys():
                verbs[category] = 1
            else:
                verbs[category] += 1

def findVerbsAndNouns(sentences):
    nouns = {}
    verbs = {}

    for sentence in sentences:
        tokens = word_tokenize(sentence)
        tags = nltk.pos_tag(tokens)
        findCategories(tokens,tags,nouns,verbs)

    X = []
    Y = []
    for noun in nouns.keys():
        X.append(noun.split('.')[1][:4])
        Y.append(nouns[noun])


    xlabel = 'noun categories'
```

```
    ylabel = 'frequency'
    title = 'Relationship between noun categories and their frequency'

    plot(X,Y,xlabel,ylabel,title)

    X = []
    Y = []
    for verb in verbs.keys():
        X.append(verb.split('.')[1][:4])
        Y.append(verbs[verb])


    xlabel = 'verb categories'
    ylabel = 'frequency'
    title = 'Relationship between verb categories and their frequency'

    plot(X,Y,xlabel,ylabel,title)


def namedEntityRecognition(sentences):
    entities = {}
    nlp = en_core_web_sm.load()
    for sentence in sentences:
        doc = nlp(sentence)
        for X in doc.ents:
```

```
            if X.label_ not in entities.keys():
                entities[X.label_] = []
            entities[X.label_].append(X.text.lower())

    return entities



def relationBetweenEntities(sentences):
    tokenized_sentences = [word_tokenize(sentence) for sentence in sentences]
    tagged_sentences = [nltk.tag.pos_tag(sentence) for sentence in tokenized_sentences]
    OF = re.compile(r'.*\bof\b.*')
    IN = re.compile(r'.*\bin\b(?!\b.+ing)')
    print('PERSON-ORGANISATION Relationships:')
    for i, sent in enumerate(tagged_sentences):
        sent = nltk.chunk.ne_chunk(sent) # ne_chunk method expects one tagged sentence
        rels = extract_rels('PER', 'ORG', sent, corpus='ace', pattern=IN, window=10)
        for rel in rels:
            print(rtuple(rel))

    print()
    print('PERSON-GPE Relationships:')
    for i, sent in enumerate(tagged_sentences):
        sent = nltk.chunk.ne_chunk(sent) # ne_chunk method expects one tagged sentence
        rels = extract_rels('PER', 'GPE', sent, corpus='ace', pattern=OF, window=10)
        for rel in rels:
```

```
            print(rtuple(rel))



def testResults1():
    results = []
    file = open('labelling.txt','r')
    lines = file.readlines()
    n = int(lines.pop(0))
    for i in range(n):
        vals = lines.pop()
        vals = vals.split('=')
        line = []
        for j in vals:
            line.append(j.replace('\n','').replace('\t',''))
        results.append(tuple(line))

    #print(results,len(results))
    return results

def testResults2():
    results = []
    file = open('labelling2.txt','r')
    lines = file.readlines()
    n = int(lines.pop(0))
    while lines:
```

```
        vals = lines.pop(0)
        vals = vals.split('=')
        line = []
        for j in vals:
            line.append(j.replace('\n','').replace('\t',''))
        results.append(tuple(line))

    #print(results,len(results))
    return results



def writeFile(T):
    file = open("lol.txt",'w')
    file.writelines(T)

def confusionMatrix(results,entities,entity):
    true_pos = 0
    false_pos = 0
    false_neg = 0
    actual = 0

    if entity in entities.keys():
        total_pos_predicted = len(entities[entity])
    else:
        total_pos_predicted = 0
```

```
total_neg_predicted = len(results)-total_pos_predicted


for line in results:
    name,e = line[0].lower(),line[1]
    if e == entity:
        actual += 1
        if entity in entities.keys():
            if name in entities[entity]:
                true_pos += 1




false_pos = total_pos_predicted-true_pos
false_neg = actual-true_pos
true_neg = total_neg_predicted-false_neg


recall = true_pos/(true_pos+false_neg)
precision = true_pos/(true_pos+false_pos)
#print(recall,precision)
fscore = 2*recall*precision/(recall+precision)


print('F-measue =',fscore)


matrix = [[true_pos,false_pos],[false_neg,true_neg]]
```

```
    df_cm = pd.DataFrame(matrix, index = [i for i in ['Positive','Negative']],
                         columns = [i for i in ['Positive','Negative']])

    xlabel = 'Actual Values'
    ylabel = 'Predicted Values'
    title = 'Confusion Matrix of '+entity
    ax = plt.subplot()
    sn.heatmap(df_cm, annot=True,ax = ax)
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
    ax.set_title(title)
    plt.show()



def testBook1():
    doc = codecs.open('testing.txt','r','utf-8')
    sentences = getSentences(doc,'','',True)
    results = testResults1()
    entities = namedEntityRecognition(sentences)

    confusionMatrix(results,entities,'PERSON')

##    tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
##    content = doc.read()
```

```
##
##    nlp = en_core_web_sm.load()
##    displacy.serve(nlp(content[1000:2000]), style='ent')
##    displacy.serve(nlp(sentences[5]), style='dep')


def testBook2():
    doc = codecs.open('testing2.txt','r','utf-8')
    sentences = getSentences(doc,'','',True)
    results = testResults2()
    entities = namedEntityRecognition(sentences)


    confusionMatrix(results,entities,'DATE')


##    tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
##    content = doc.read()
##    #print(content)


    nlp = en_core_web_sm.load()
    #displacy.serve(nlp(content[:900]), style='ent')
    displacy.serve(nlp(sentences[5]), style='dep')


    #print(entities)


def NER(sentences):
    entities = namedEntityRecognition(sentences)
```

```
    X = []
    Y = []
    for i in entities.keys():
        X.append(i[:4])
        Y.append(len(entities[i]))

    xlabel = 'entities'
    ylabel = 'frequency'
    title = 'Relationship between entities and their frequency'

    plot(X,Y,xlabel,ylabel,title)

def book1Tasks():
    sentences = preprocessBook1()
    #tokens,tokenized_sentences = tokenize(sentences)
    #makeWordCloud(tokens)
    #plotRelationShip(tokens)
    #tokens_without_stopwords = removeStopWords(tokens)
    #plotRelationShip(tokens_without_stopwords)
    #plotTop20Words(tokens)
    #PoSTagging(tokenized_sentences)
    #findVerbsAndNouns(sentences)
    #NER(sentences)
    relationBetweenEntities(sentences)
    #testBook1()
```

```
def book2Tasks():
##    sentences = preprocessBook2()
##    tokens,tokenized_sentences = tokenize(sentences)
##    makeWordCloud(tokens)
##    plotRelationShip(tokens)
##    tokens_without_stopwords = removeStopWords(tokens)
##    plotRelationShip(tokens_without_stopwords)
##    plotTop20Words(tokens)
##    PoSTagging(tokenized_sentences)
##    findVerbsAndNouns(sentences)
##    NER(sentences)
    #relationBetweenEntities(sentences)
    testBook2()


def main():
    book1Tasks()
    #book2Tasks()



main()
```