NLP PROJECT

NLP ROUND 1:

TEAM NAME:

STARKS

TEAM MEMBERS:

KUBER VARSHNEY 19UCS194

SAATVIK SINGH 19UCS133

SAMARPIT SACHAN 19UCS036

Data Description:

Books used:

BOOK 1: Williwaw: A Novel by Gore Vidal.

BOOK 2: Hans Andersen's Fairy Tales by H. C. Andersen.

BOOK 3: Greensea Island by Victor Bridges.

Code Link:

https://github.com/kubercodes/NLP-Project/blob/main/NLP_Project_2.ipynb

PROBLEM STATEMENT:

1. Import the text, let us call it as T1 and T2 (books that you have downloaded)

```
with open("/content/book1.txt") as f1:
  T1 = f1.read()
```

```
with open("/content/book2.txt") as f2:
  T2 = f2.read()
```

2. Perform simple text pre-processing steps and tokenize the text T1 and T2 — you may have to do the removal of running section / chapter names and so on.

- Tokenization

Here we are tokenizing the two books where we are splitting it into individual terms. Following are the codes snippets for tokenizing the books.

```
# Tokenizing T1
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize
text = open('book1.txt',mode='r',encoding='utf-8').read()
T1_token = token = word_tokenize(text)
T1_token
T2_token = token = word_tokenize(text)
T2_token
```

Tokenisation of Book 1:

First 10 tokens:

```
'\ufeffThe',
'Project',
'Gutenberg',
'eBook',
'of',
'Williwaw',
',',
'by',
'Gore',
'Vidal',
```

Last 10 tokens:

```
'',',
'll',
'go',
'below',
'now',
',',
'"',
'said',
'Evans',
'.',
```

Tokenisation of Book 2:

First 10 tokens:

```
'\ufeffThe',
'Project',
'Gutenberg',
'eBook',
'of',
'Hans',
'Andersen',
"'s",
'Fairy',
'Tales',
```

Last 10 tokens:

```
'said',
'the',
'king',
'253',
'Their',
'slippers',
'flew',
'about',
'their',
'ears',
```

- Remove headings and unwanted lines:

Lines starting with a new line or special characters are being removed in this piece of code.

For example:

```
_A Novel_

                        *       *       *       *       *


_Chapter One_
```

Code snippet:

```
# Here we are removing the headings and new lines.
# Lines starting with _ , * , [ and further more are being removed
for line in f.readlines():
  line = line.replace('\n',' ')         # removing blank lines
  #line = line.replace('',' ')
  l_list = []
  l_list = line.split(" ")
  if not (line.startswith('_') or line.startswith('       *') or line.startswith('[')
   or line.startswith('*') or line.startswith('ii') or l_list[0] == '\n' or l_list[0] == 'i\n'):
    f1.write(line)
f.close()
f1.close()
```

- Remove the punctuation marks:

Now we clean the different types of punctuation marks present in the text so that we are only left with letters.

Code snippet:

```
# Removing the punctuation marks and
# Removing ’ , ” , “ , ‘
for s in f.readlines():
  for char in string.punctuation:
    s = s.replace(char, ' ')
    s = s.replace("’","")
    s = s.replace("”","")
    s = s.replace("“","")
    s = s.replace("‘","")
```

- Remove chapter headings:

Chapter headings in capital letters are removed and cleaned here. It is one of the ways to get rid of the running sections in the book.

For example:

```
THE MARSH KING’S DAUGHTER
```

Code snippet:

```
def onlyUpper(word):
    for c in word:
        if not c.isupper():
            return False
    return True
```

```
# removing chapter headings as they are in capital letters
for w in words:
  if not onlyUpper(w):
    good_words.append(w)
```

- Dealing with Uppercase alphabets.

Uppercase and lowercase have different meanings so we need to change the uppercase alphabet into lowercase alphabets like The -> the or from It -> it and such many other examples.

```
# converting capital letter alphabet from words to lower case like The -> the
  result = ""
  for w in good_words:
    w = w.lower()
    result = result + w + " "

  f1.write(result)
```

3. Analyse the frequency distribution of tokens in T1 and T2 separately.

Code snippet from book 1:

```
from collections import Counter
with open("/content/book1_1.txt") as f1:
  T1 = f1.read()
T1_list = T1.split(" ")
counts = Counter(T1_list)
print(counts)
```

Code snippet from book 2:

```
from collections import Counter
with open("/content/book2_1.txt") as f2:
  T2 = f2.read()
T2_list = T2.split(" ")
counts = Counter(T2_list)
print(counts)
```

T1 frequency counts:

```
Counter({'the': 3977, 'he': 1606, 'to': 1320, 'and': 1204, 'was': 1058, 'of': 1023,
```

T2 frequency counts:

```
Counter({'the': 5216, 'and': 2787, 'to': 1575, 'of': 1512, 'a': 1291, 'her': 1095,
```

4. Create a Word Cloud of T1 and T2 using the token that you have got.

Word Clouds are visual representations of words that give greater prominence to words that appear more frequently.

Code snippet for book 1:

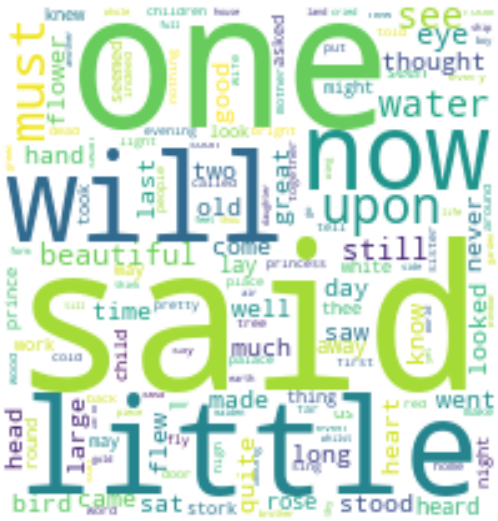
```
# Creating Word Cloud
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS
import sys,os
text = open('book1_1.txt',mode='r',encoding='utf-8').read()
wc = WordCloud(
    background_color='white',
    height = 200,
    width=200
)
wc.generate(text)
wc.to_file('wordcloud_1.png')
```

Code snippet for book 2:

```
# Creating Word Cloud
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS
import sys,os
text = open('book2_1.txt',mode='r',encoding='utf-8').read()
wc = WordCloud(
    background_color='white',
    height = 200,
    width=200
)
wc.generate(text)
wc.to_file('wordcloud_2.png')
```

T1 word cloud:

T2 word cloud:



5. Remove the stop words from T1 and T2 and then again create a word cloud.

Stopwords are the words in any language which does not add much meaning to a sentence. They can safely be ignored without sacrificing the meaning of the sentence.

Packages used: We have used **NTLK** library of python to help us remove the stopwords from our text data.

```
# Importing stopwords from the nltk library
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
print(stopwords.words('english'))
```

```
# Removing stopwords
word_token = word_tokenize(text)
filter_word = []
for w in word_token:
  if w not in stop_word:
    f1.write(w)
    f1.write(" ")
f1.close()
```

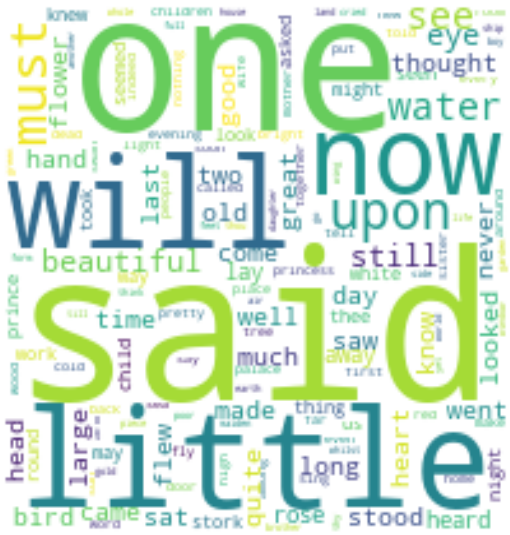
Some examples of Stopwords:

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you'

'you', "you're", "you've", "you'll", "you'd", 'your', 'yours'
```

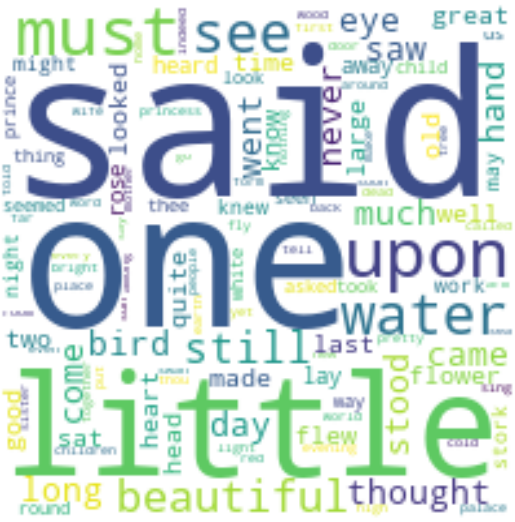
T1 word cloud before removing stop words:

T1 word cloud after removing stop words:

T2 word cloud before removing stop words:



T2 word cloud after removing stop words:



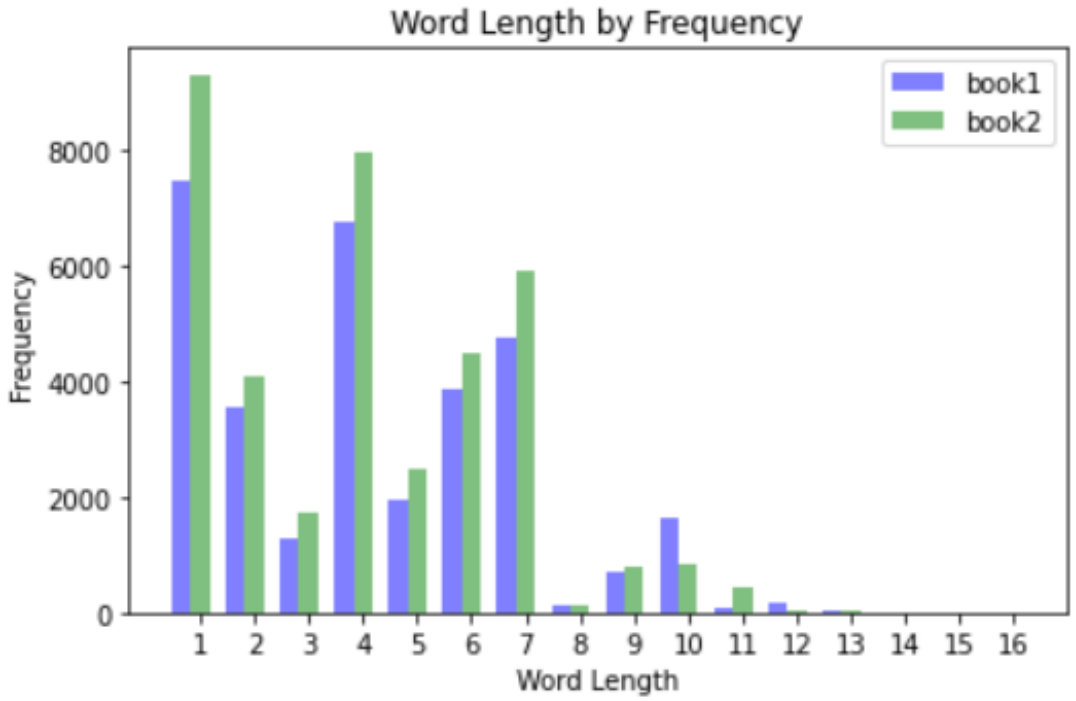
6. Evaluate the relationship between the word length and frequency for both T1 and T2.

Code Snippets:

```
text = open('book1_2.txt',mode='r',encoding='utf-8').read()
word_token = word_tokenize(text)
dic1 = {}
for char in word_token:
  #if(len(char) == 14):
     # print(char)
  if dic1.get(len(char),None) == None:
    dic1[len(char)] = 1
  else:
    dic1[len(char)] += 1
```

```
text = open('book2_2.txt',mode='r',encoding='utf-8').read()
word_token = word_tokenize(text)
dic2 = {}
for char in word_token:
  #if(len(char) == 16):
    #print(char)
  if dic2.get(len(char),None) == None:
    dic2[len(char)] = 1
  else:
    dic2[len(char)] += 1
```

Comparing word length and frequency of both books.



7. Do PoS Tagging for both T1 and T2 using anyone of the four tag sets studied in the class and get the distribution of various tags.

Here we mark each word with their part of speech assigning them different tag sets.

```
# POS TAGGING in BOOK 1
from nltk.corpus import brown
brown_tagged_sents = brown.tagged_sents(categories='news')
brown_sents = brown.sents(categories='news')
unigram_tagger = nltk.UnigramTagger(brown_tagged_sents)
text = open('book1_2.txt',mode='r',encoding='utf-8').read()
word_token = word_tokenize(text)
unigram_tagger.tag(word_token)
```

```
# POS TAGGING in BOOK 2
from nltk.corpus import brown
brown_tagged_sents = brown.tagged_sents(categories='news')
brown_sents = brown.sents(categories='news')
unigram_tagger = nltk.UnigramTagger(brown_tagged_sents)
text = open('book2_2.txt',mode='r',encoding='utf-8').read()
word_token = word_tokenize(text)
unigram_tagger.tag(word_token)
```

BOOK 1:

Output snippets:

```
('Project', None),
('Gutenberg', None),
('eBook', None),
('Williwaw', None),
('Gore', 'NP'),
('Vidal', None),
('This', 'DT'),
('eBook', None),
('use', 'VB'),
('anyone', 'PN'),
('anywhere', 'RB'),
```

```
('Library', 'NN-TL'),
('By', 'IN'),
('1946', 'CD'),
('writing', 'VBG'),
('publisher', None),
('except', 'IN'),
('reviewer', None),
('wishes', None),
('quote', 'NN'),
('brief', 'JJ'),
('passages', 'NNS'),
```

BOOK 2:

Output snippets:

```
('Project', None),
('Gutenberg', None),
('eBook', None),
('Hans', None),
('Andersen', None),
('Fairy', 'NN-TL'),
('Tales', None),
('Hans', None),
('Andersen', None),
('This', 'DT'),
('eBook', None),
('use', 'VB'),
```

```
('entered', 'VBD'),
('large', 'JJ'),
('cold', 'JJ'),
('empty', None),
('hall', 'NN'),
('Tailpiece', None),
('The', 'AT'),
('elfin', None),
('kings', None),
('housekeeper', None),
('The', 'AT'),
('mer', None),
('king', 'NN'),
```

NLP ROUND 2:

PROBLEM STATEMENT:

First Part:

1. Find the nouns and verbs in both the novels. Get the immediate categories (parent) that these words fall under in the WordNet.

BOOK1:

List of Nouns: (Output provided)

```
list1 = []
for item in tags1:
    if (item[1] in {'NN','NNP','NNS'}):
        list1.append(item)
        print(item[0])
```

```
shot
nose
eyes
neck
mile
gear
wheel
crew
bed
noon
door
men
wheel
course
memory
courses
heart
entrance
speed
navy
detachment
point
boats
speed
```

```
print("Total number of Nouns in book1 are : "+ str(len(list1)))
```

```
Total number of Nouns in book1 are : 6076
```

List of Verbs: (Output provided)

```
list1 = []
for item in tags1:
    if (item[1] in {'VB','VBP','VBG','VBD','VBN','VBZ'}):
        list1.append(item)
        print(item[0])
```

```
looked
wind
floated
came
stood
sailing
calm
thought
seem
expected
tell
came
stood
sailing
calm
thought
seem
expected
tell
came
ask
interested
pointed
seems
```

```
print("Total number of Verbs in book1 are : "+ str(len(list1)))
```

```
Total number of Verbs in book1 are : 7249
```

BOOK2:

List of Nouns: (Output provided)

```
list1 = []
for item in tags2:
    if (item[1] in {'NN','NNP','NNS'}):
        list1.append(item)
        print(item[0])
```

```
arrival
milk
backs
room
woman
advice
journey
home
woman
clothes
party
country
green
woman
farewell
birds
day
songs
trees
forest
foliage
green
horse
coach
```

```
print("Total number of Nouns in book2 are : "+ str(len(list1)))
```

```
Total number of Nouns in book2 are : 8564
```

List of Verbs: (Output Provided)

```
list1 = []
for item in tags2:
    if (item[1] in {'VB','VBP','VBG','VBD','VBN','VBZ'}):
        list1.append(item)
        print(item[0])
```

```
led
grew
compared
bounded
delight
played
set
stuffed
prepared
played
spent
knew
seemed
wanting
tell
sitting
looking
painted
rose
painted
rose
made
sink
```

```
print("Total number of Verbs in book2 are : "+ str(len(list1)))
```

```
Total number of Verbs in book2 are : 7291
```

Packages Used: wordnet

WordNET is a lexical database of words in more than 200 languages in which we have adjectives, adverbs, nouns, and verbs grouped differently into a set of cognitive synonyms, where each word in the database is expressing its distinct concept. The cognitive synonyms which are called synsets are presented in the database with lexical and semantic relations.

Immediate categories (parent) that these words fall under in the WordNet.

Book1:

```
list2 = {}
from nltk.corpus import wordnet

for item in list1:
    cat = wordnet.synsets(item[0])
    for i in cat:
        if i.lexname()[0] == 'n':
            if i.lexname() in list2:
                list2[i.lexname()]+=1
            else:
                list2[i.lexname()]=1
```

```
print('Categories of Nouns in Wordnet and their respective frequency is given below-.. (for book1)')
print()
print(list2)
print()
print('Total categories in Nouns in the wordnet are : ' + str(len(list2)))
```

```
Categories of Nouns in Wordnet and their respective frequency is given below-.. (for book1)

{'noun.act': 3458, 'noun.cognition': 3173, 'noun.location': 1915, 'noun.Tops': 375, 'noun.group': 2626,

Total categories in Nouns in the wordnet are : 26
```

```
list2 = {}
from nltk.corpus import wordnet

for item in list1:
    cat = wordnet.synsets(item[0])
    for i in cat:
        if i.lexname()[0] == 'v':
            if i.lexname() in list2:
                list2[i.lexname()]+=1
            else:
                list2[i.lexname()]=1
```

```
print('Categories of Verbs in Wordnet and their respective frequency is given below.. (for book1)')
print()
print(list2)
print()
print('Total categoris in Verbs in the wordnet are : ' + str(len(list2)))
```

```
Categories of Verbs in Wordnet and their respective frequency is given below.. (for book1)

{'verb.consumption': 1899, 'verb.social': 7130, 'verb.possession': 6013, 'verb.communication': 13531,

Total categoris in Verbs in the wordnet are : 15
```

Book2:

```
list2 = {}
from nltk.corpus import wordnet

for item in list1:
    cat = wordnet.synsets(item[0])
    for i in cat:
        if i.lexname()[0] == 'n':
            if i.lexname() in list2:
                list2[i.lexname()]+=1
            else:
                list2[i.lexname()]=1
print('Categories of Nouns in Wordnet and their respective frequency is given below-.. (for book2)')
print()
print(list2)
print()
print('Total categories in Nouns in the wordnet are : ' + str(len(list2)))
```

```
Categories of Nouns in Wordnet and their respective frequency is given below-.. (for book2)

{'noun.location': 3156, 'noun.relation': 145, 'noun.artifact': 9131, 'noun.object': 2218, 'noun.cognition': 3712,

Total categories in Nouns in the wordnet are : 26
```

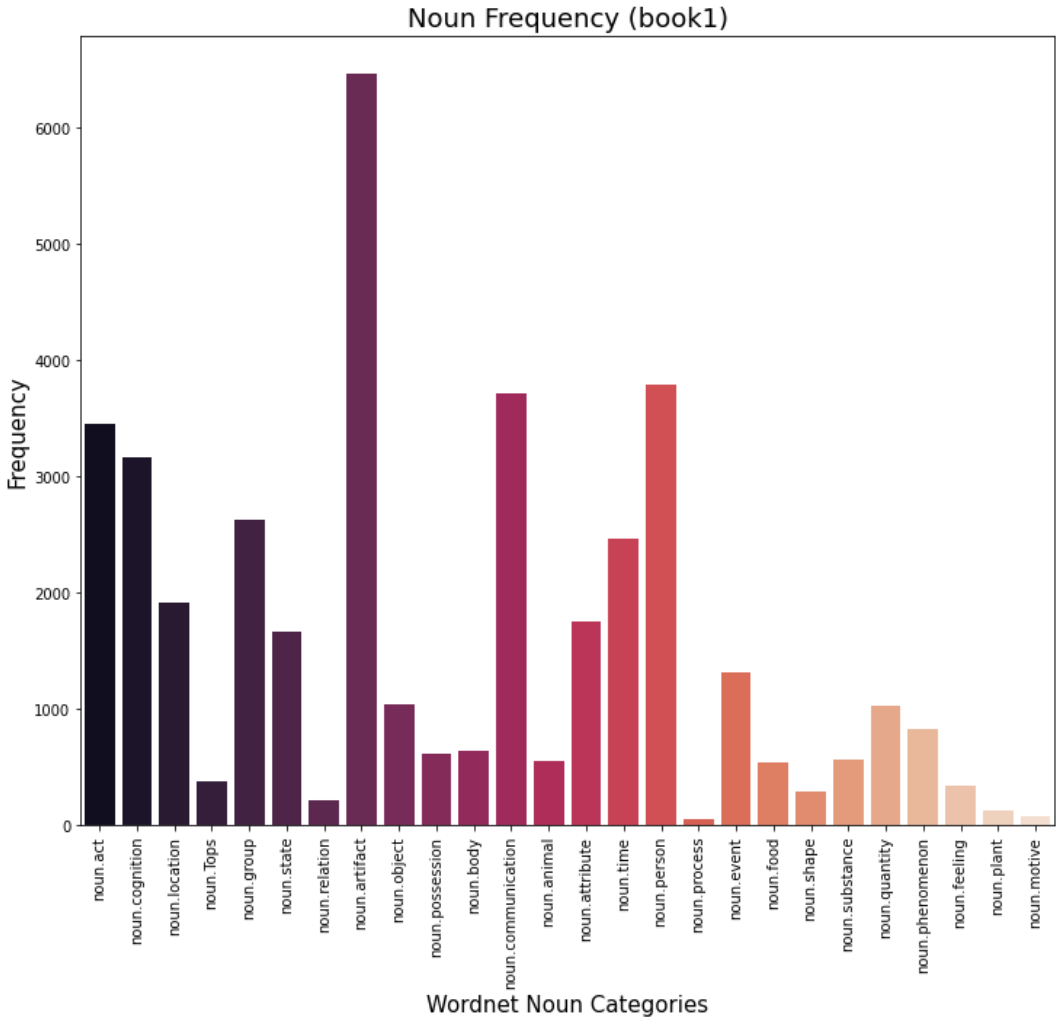
```
list2 = {}
from nltk.corpus import wordnet

for item in list1:
    cat = wordnet.synsets(item[0])
    for i in cat:
        if i.lexname()[0] == 'v':
            if i.lexname() in list2:
                list2[i.lexname()]+=1
            else:
                list2[i.lexname()]=1
print('Categories of Verbs in Wordnet and their respective frequency is given below.. (for book2)')
print()
print(list2)
print()
print('Total categories in Verbs in the wordnet are : ' + str(len(list2)))
```

```
Categories of Verbs in Wordnet and their respective frequency is given below.. (for book2)

{'verb.consumption': 1678, 'verb.social': 8146, 'verb.possession': 6656, 'verb.communication': 14057,

Total categories in Verbs in the wordnet are : 15
```

2. Get the frequency of each category for each noun and verb in their corresponding hierarchies and plot a histogram for the same for each novel.

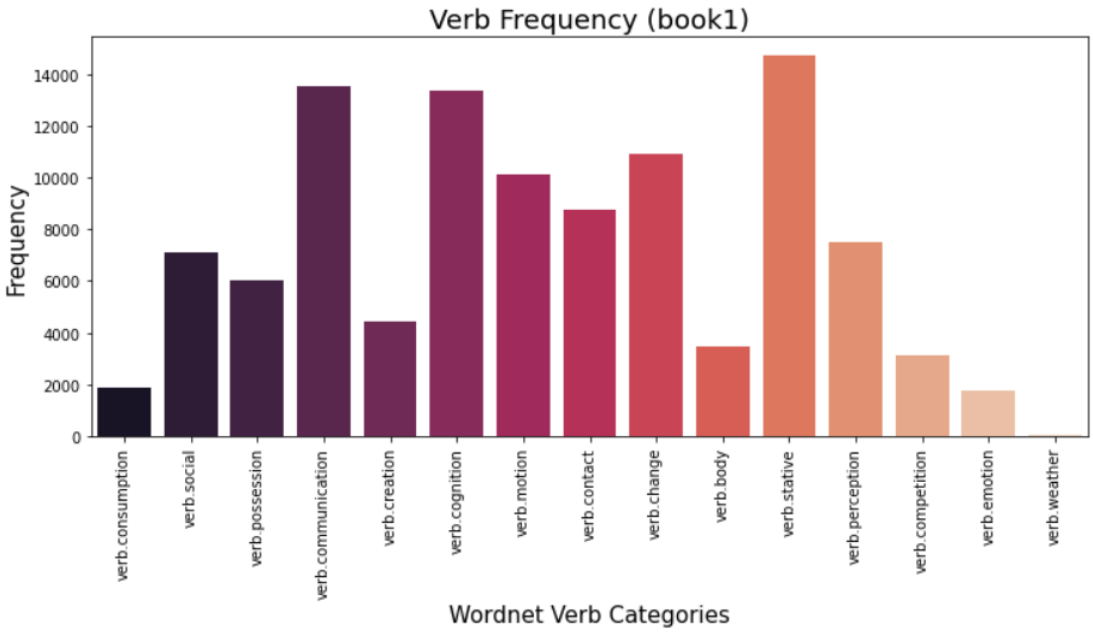
BOOK1 (NOUN FREQUENCY):

```
# histogram for noun frequency of book1
import seaborn as sns
X = list(list2.keys())
y = list(list2.values())
plt.figure(figsize=(12,10))
plt.xticks(rotation=90)
plt.title('Noun Frequency (book1)',size=18)
plt.xlabel('Wordnet Noun Categories',size=15)
plt.ylabel('Frequency',size=15)
sns.barplot(X,y,palette='rocket')
```



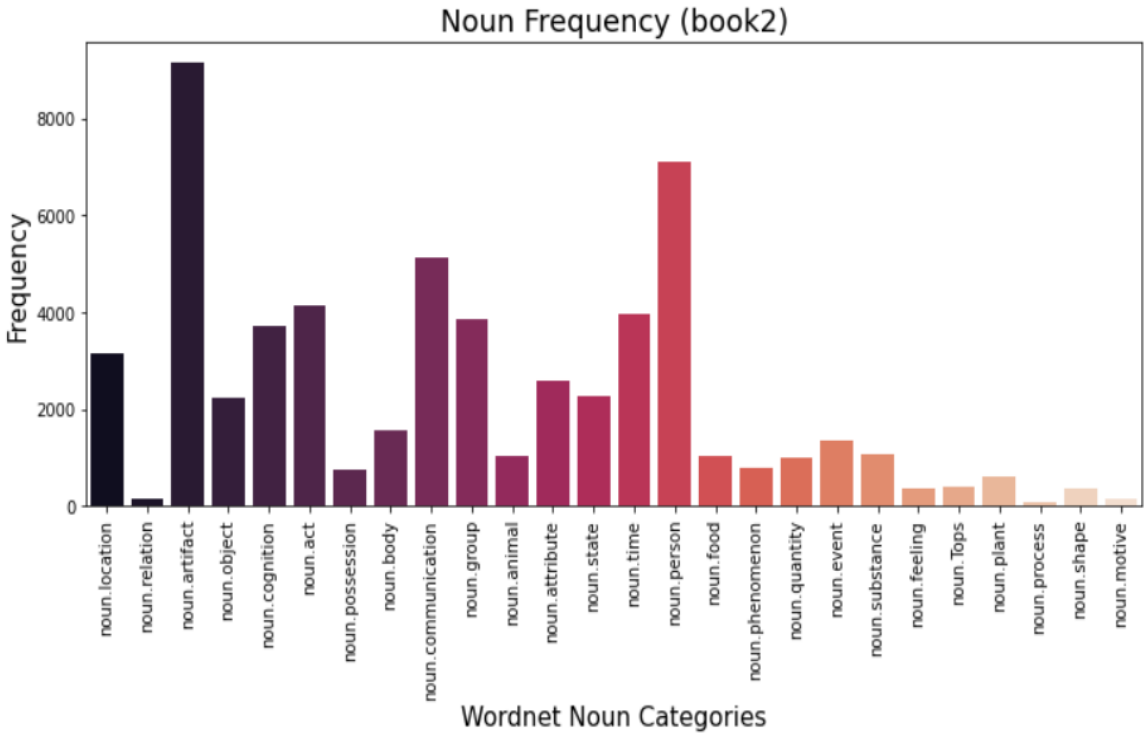
BOOK1 (VERB FREQUENCY):

```
#histogram for verb frequency of book1
X = list(list2.keys())
y = list(list2.values())
plt.figure(figsize=(12,5))
plt.xticks(rotation=90)
plt.title('Verb Frequency (book1)',size=18)
plt.xlabel('Wordnet Verb Categories',size=15)
plt.ylabel('Frequency',size=15)
sns.barplot(X,y,palette='rocket')
```



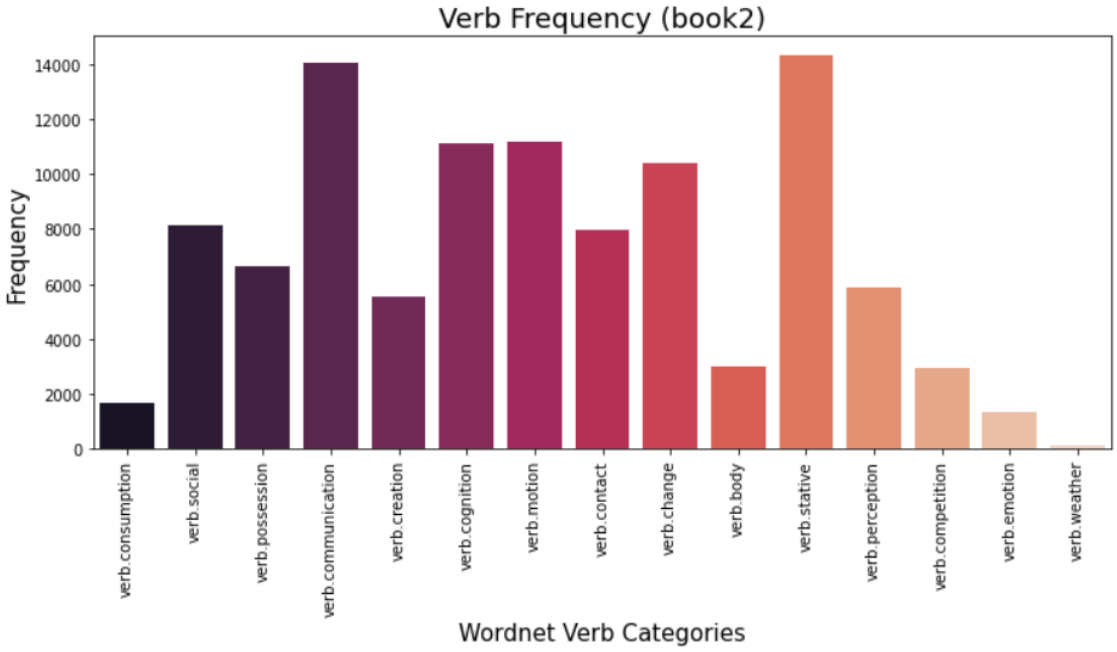
BOOK2 (NOUN FREQUENCY):

```
# histogram for Noun Frequency of Book2
X = list(list2.keys())
y = list(list2.values())
plt.figure(figsize=(12,5))
plt.xticks(rotation=90)
plt.title('Noun Frequency (book2)',size=18)
plt.xlabel('Wordnet Noun Categories',size=15)
plt.ylabel('Frequency',size=15)
sns.barplot(X,y,palette='rocket')
```



BOOK 2 (VERB FREQUENCY):

```
#histogram for Verb frequency of book2
X = list(list2.keys())
y = list(list2.values())
plt.figure(figsize=(12,5))
plt.xticks(rotation=90)
plt.title('Verb Frequency (book2)',size=18)
plt.xlabel('Wordnet Verb Categories',size=15)
plt.ylabel('Frequency',size=15)
sns.barplot(X,y,palette='rocket')
```



Second Part:

1. Recognise all Persons, Location, Organisation (Types given in Fig 22.1) in book. For this you have to do two steps: (1) First recognise all the entity and then (2) recognise all entity types. Use performance measures to measure the performance of the method used - For evaluation you take a considerable amount of random passages from the Novel, do a manual labelling and then compare your result with it. Present the accuracy with F1 score here.

Finding all entities and labelling them with the appropriate entity types.

Package used: spaCy

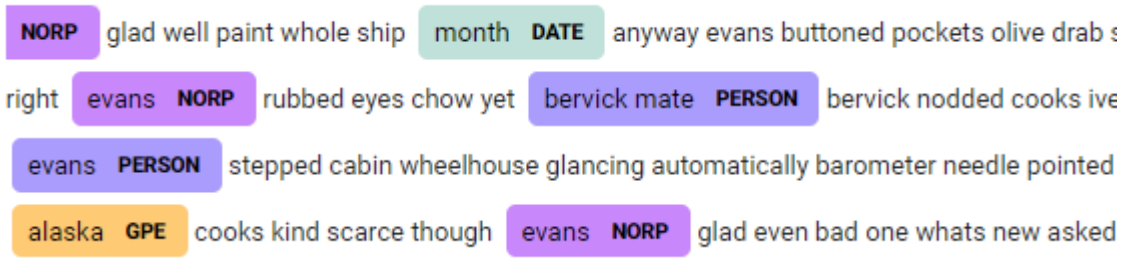
spaCy acts as a one-stop-shop for various tasks used in NLP projects, such as Tokenization, Lemmatisation, Part-of-speech (POS) tagging, Entity recognition, Dependency parsing, Sentence recognition, Word-to-vector transformations, and other cleaning and normalization text methods.

Book1:

```
import spacy
from spacy import displacy
NER = spacy.load("en_core_web_sm")
text = open('book1_2.txt',mode='r',encoding='utf-8').read()
text1 = NER(text)
for word in text1.ents:
    print(word.text,word.label_)
```

```
paragraph 1 1
nonproprietary
gutenberg tm
gutenberg tm license
gutenberg tm
gutenberg tm
60 days
section 4
30 days
90 days
gutenberg tm
```

NER LABELLING OF BOOK1:



Book2:

```
import spacy
from spacy import displacy
NER = spacy.load("en_core_web_sm")
text = open('book2_2.txt',mode='r',encoding='utf-8').read()
text2 = NER(text)
for word in text2.ents:
    print(word.text)
```

```
robber maiden
hall elfin
first
thee
mermaid
evening
sun arose
one
city palace garden
two
one
thee
bubbles
two
one day
mans
```

NER LABELLING OF BOOK2:

air soared high rosy cloud bring two CARDINAL little ones brother sister began nightingale sing peasants wife sat

ng day upon roof palace expecting jumped old mans NORP lap danced around floor storks tell young ones ever ma

ast tale two CARDINAL oldest longest tales told among storks one CARDINAL know placed mother ark waters fo

hundreds years DATE turn told better till telling best first ORDINAL pair storks knew summer quarters DATE

Use B1, B2 and B3 for the following:

Third Part:

1. Create TF-IDF vectors for all books and find the cosine similarity between each of them and find which two books are more similar.

Packages Used: Count Vectorizer

Count Vectorizer is used to transform a given text into a vector on the basis of the frequency (count) of each word that occurs in the entire text. This is helpful when we have multiple such texts, and we wish to convert each word in each text into vectors.

TF-IDF vectors of each book:

```
# converting books into tf-idf vectors
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from nltk.corpus import stopwords
import numpy as np
import numpy.linalg as LA
text1 = open('book1_2.txt',mode='r',encoding='utf-8').read()
text2 = open('book2_2.txt',mode='r',encoding='utf-8').read()
text3 = open('book3_2.txt',mode='r',encoding='utf-8').read()
train_set = [text1,text2,text3]  # Documents

stopWords = stopwords.words('english')

vectorizer = CountVectorizer(stop_words = stopWords)

transformer = TfidfTransformer()


trainVectorizerArray = vectorizer.fit_transform(train_set).toarray()

transformer.fit(trainVectorizerArray)
lists =  (transformer.transform(trainVectorizerArray).toarray())
lists
```

Packages used: TfidfTransformer , numpy.linalg

Tf-Idf Transformer transforms a count matrix to a normalized tf or tf-idf representation.

Tf means term-frequency while tf-idf means term-frequency times inverse document-frequency. This is a common term weighting scheme in information retrieval, that has also found good use in document classification.

NumPy package contains **numpy.linalg** module that provides all the functionality required for linear algebra. For example: dot product, inverse of a matrix and so on.

Representation of TF-IDF vectors of each book:

BOOK1:

```
lists[0][0:50]                                    # first 50 tf-idf vectors of book1
```

```
array([0.00068328, 0.        , 0.00068328, 0.        , 0.        ,
       0.00068328, 0.        , 0.        , 0.00089843, 0.00068328,
       0.00068328, 0.        , 0.00068328, 0.00068328, 0.00068328,
       0.00068328, 0.00068328, 0.00068328, 0.00068328, 0.00089843,
       0.        , 0.00068328, 0.00068328, 0.00068328, 0.00136655,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.00089843, 0.00449213, 0.00053062, 0.        , 0.        ,
       0.        , 0.01114312, 0.        , 0.01024914, 0.        ,
       0.        , 0.00068328, 0.00478293, 0.        , 0.        ,
       0.        , 0.00539055, 0.        , 0.        , 0.00068328])
```

BOOK2:

```
lists[1][0:50]                                    # first 50 tf-idf vectors of book2
```

```
array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.00096605, 0.        ,
       0.        , 0.        , 0.00150046, 0.00381074, 0.        ,
       0.        , 0.01575478, 0.        , 0.        , 0.01016197,
       0.00289816, 0.        , 0.        , 0.        , 0.00193211,
       0.        , 0.        , 0.        , 0.00193211, 0.        ])
```

BOOK3:

```
lists[2][0:50]                                    # first 50 tf-idf vectors of book3
```

```
array([0.00073161, 0.00288593, 0.00073161, 0.00096198, 0.00096198,
       0.00073161, 0.00096198, 0.00096198, 0.        , 0.00073161,
       0.00073161, 0.00096198, 0.00073161, 0.00073161, 0.00073161,
       0.00073161, 0.00073161, 0.00073161, 0.00073161, 0.        ,
       0.00096198, 0.00073161, 0.00073161, 0.00073161, 0.00146321,
       0.00096198, 0.00096198, 0.00288593, 0.00292643, 0.00096198,
       0.        , 0.        , 0.00056816, 0.        , 0.00096198,
       0.00096198, 0.02215818, 0.00096198, 0.00219482, 0.        ,
       0.00073161, 0.00292643, 0.00365804, 0.0038479 , 0.00365804,
       0.00192395, 0.        , 0.00769581, 0.01243733, 0.00073161])
```

Cosine Similarity of each book:

Output from the code:

The less the angle between them that is the less the result, the more similar they are to each other.

```
[ ] result = dot(lists[0],lists[1])/(norm(lists[0])*norm(lists[1]))          # cosine similarity of book1-book2
    print(result)
```

```
0.3795863611830241
```

```
[ ] result = dot(lists[1],lists[2])/(norm(lists[1])*norm(lists[2]))          # cosine similarity of book2-book3
    print(result)
```

```
0.5984607970577177
```

```
[ ] result = dot(lists[0],lists[2])/(norm(lists[0])*norm(lists[2]))          # cosine similarity of book1-book3
    print(result)
```

```
0.4189852600452222
```

Book1 and Book 2 are more similar to each other compared to the rest.

2. Do lemmatization of the books and recreate the TF-IDF vectors for all the books and find the cosine similarity of each pair of books.

Lemmatization is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item. Lemmatization is similar to stemming but it brings context to the words. So, it links words with similar meanings to one word.

Package used: WordNetLemmatizer

Lemmatize using WordNet's built-in morphy function. Returns the input word unchanged if it cannot be found in WordNet.

Lemmatization of each book:

```
#lemmatization for book1
import nltk
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

# a denotes adjective in "pos"
f1=open("book1_2.txt",'r')
f1_l=open("book1_2_l.txt",'w')
for line in f1.readlines():
  for words in line.split():
    wl = lemmatizer.lemmatize(words)
    f1_l.write(wl+' ')
```

```
#lemmatization for book2
import nltk
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

# a denotes adjective in "pos"
f1=open("book2_2.txt",'r')
f1_l=open("book2_2_l.txt",'w')
for line in f1.readlines():
  for words in line.split():
    wl = lemmatizer.lemmatize(words)
    f1_l.write(wl+' ')
```

```
#lemmatization for book3
import nltk
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

# a denotes adjective in "pos"
f1=open("book3_2.txt",'r')
f1_l=open("book3_2_l.txt",'w')
for line in f1.readlines():
  for words in line.split():
    wl = lemmatizer.lemmatize(words)
    f1_l.write(wl+' ')
  f1_l.write('\n')
```

Recreating TF-IDF vectors after Lemmatization:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from nltk.corpus import stopwords
import numpy as np
import numpy.linalg as LA
text1 = open('book1_2_l.txt',mode='r',encoding='utf-8').read()
text2 = open('book2_2_l.txt',mode='r',encoding='utf-8').read()
text3 = open('book3_2_l.txt',mode='r',encoding='utf-8').read()
train_set = [text1,text2,text3]  # Documents

stopWords = stopwords.words('english')

vectorizer = CountVectorizer(stop_words = stopWords)

transformer = TfidfTransformer()


trainVectorizerArray = vectorizer.fit_transform(train_set).toarray()

transformer.fit(trainVectorizerArray)
lists =  (transformer.transform(trainVectorizerArray).toarray())
lists.shape
```

```
(3, 9831)
```

Cosine Similarity between different books after lemmatization:

```
result = dot(lists[0],lists[1])/(norm(lists[0])*norm(lists[1]))          # cosine similarity of book1-book2
print(result)
```

```
0.3820874831090321
```

```
result = dot(lists[1],lists[2])/(norm(lists[1])*norm(lists[2]))          # cosine similarity of book2-book3
print(result)
```

```
0.597744984679492
```

```
result = dot(lists[0],lists[2])/(norm(lists[0])*norm(lists[2]))          # cosine similarity of book1-book3
print(result)
```

```
0.4286115756408054
```

Book1 and Book 2 are more similar to each other compared to the rest after lemmatization.