CMPE 259: Homework 1: Regular expressions, text normalization, and edit distance

The parts that you need to complete are marked as Exercises.

Part 0: Initialization & Setup

In [1]:
```
# importing required libraries
import re
import nltk
from nltk.corpus import movie_reviews
import string
import pandas as pd
from nltk.corpus import stopwords

nltk.download('movie_reviews')
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package movie_reviews to /root/nltk_data...
[nltk_data]   Unzipping corpora/movie_reviews.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
```

Out[1]:
```
True
```

Part 1: Regular Expressions

Extracting license plate numbers, IDs, emails and mailing addresses from a document

Document creation

In [2]:
```
sentence = 'I am 20 years old. My previous license plate number was 4XUI302 and my new o
sentence
```

Out[2]:
```
'I am 20 years old. My previous license plate number was 4XUI302 and my new one is 3A-27
8. My ID is J987492 and my address is 123 Main street, San Jose, CA. Please email me at
myemail123+spam@google.cg or jane.doe@sjsu.edu'
```

Extracting license plate numbers

In [3]:
```
# The format of license plate number is a digit then 2 or 3 letters (one of which can be

regex = re.compile(r'(\d{1}[A-Za-z-]{2,3}\d{3})')
lincense_plate_numbers = regex.findall(sentence)
lincense_plate_numbers
```

Out[3]:
```
['4XUI302', '3A-278']
```

Exercise 1-1: Extract the ID numbers from the document.

In [4]:
```
# The format of the IDs is one character/letter and then 6 digits
regex = re.compile(r'([A-Za-z]\d{6})')
ids = regex.findall(sentence)
ids
```
Out[4]:
```
['J987492']
```

Exercise 1-2: Extract the email IDs from the document

In [5]:
```
regex = re.compile(r'([\d\w+.]*[@]+[\w\d]+[.]+[a-z]*)')
emails = regex.findall(sentence)
emails
```
Out[5]:
```
['myemail123+spam@google.cg', 'jane.doe@sjsu.edu']
```

Exercise 1-3: Extract the mailing address from the document

In [6]:
```
# Starts with 3 digit number, then some letters, a comma, letters, a comma
regex = re.compile(r'([0-9 ]{3}[A-Za-z ]*[,]{1}[A-Za-z0-9 ]*[,]{1}[\w ]*)')
mailing_address = regex.findall(sentence)
mailing_address
```
Out[6]:
```
['123 Main street, San Jose, CA']
```

Exercise 1-4: Anonymize the license plate numbers by replacing them with the text "LP_NUM"

The re.sub function is described here: https://docs.python.org/3/library/re.html

In [9]:
```
# Now replacing license plate numbers with the string "LP_NUM"
sentence_modified = sentence
for i in lincense_plate_numbers:
  sentence_modified = sentence_modified.replace(i, "LP_NUM")
sentence_modified
```
Out[9]:
```
'I am 20 years old. My previous license plate number was LP_NUM and my new one is LP_NU
M. My ID is J987492 and my address is 123 Main street, San Jose, CA. Please email me at
myemail123+spam@google.cg or jane.doe@sjsu.edu'
```

Exercise 1-5: Replace the ID numbers with the text "ID_NUM"

In [10]:
```
for i in ids:
  sentence_modified = sentence_modified.replace(i, "ID_NUM")
sentence_modified
```
Out[10]:
```
'I am 20 years old. My previous license plate number was LP_NUM and my new one is LP_NU
M. My ID is ID_NUM and my address is 123 Main street, San Jose, CA. Please email me at m
yemail123+spam@google.cg or jane.doe@sjsu.edu'
```

Part 2: Text Processing

Count the number of words in the movie_reviews dataset (dataset uploaded in the beginning of this notebook under "Part 0: Initialization and Setup")

In [11]:
```
# print number of words in the movie review dataset
len(movie_reviews.words())
```

Out[11]: 1583820

Load the standard list of punctuation marks

In [12]:
```
punctuations = string.punctuation
punctuations
```

Out[12]:
```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

Remove punctation from movie reviews

In [13]:
```
words_wo_puncts = [x for x in movie_reviews.words() if x not in punctuations]
len(words_wo_puncts)
```

Out[13]: 1338788

Count the number of unique words

In [14]:
```
unique_words = set(words_wo_puncts)
len(unique_words)
```

Out[14]: 39737

Find the 20 most frequent words in the dataset

In [15]:
```
# top 20 highest freq words
pd.Series(words_wo_puncts).value_counts()[:20]
```

Out[15]:

	count
the	76529
a	38106
and	35576
of	34123
to	31937
is	25195
in	21822
s	18513
it	16107
that	15924
as	11378
with	10792
for	9961
his	9587
this	9578
film	9517
i	8889
he	8864
but	8634
on	7385

dtype: int64

Load the standard list of stopwords

In [16]:
```
# getting english stopwords
eng_stopwords = stopwords.words('english')
eng_stopwords
```

Out[16]:
```
['i',
 'me',
 'my',
 'myself',
 'we',
 'our',
 'ours',
 'ourselves',
 'you',
 "you're",
 "you've",
 "you'll",
 "you'd",
 'your',
 'yours',
 'yourself',
 'yourselves',
 'he',
 'him',
 'his',
 'himself',
 'she',
 "she's",
 'her',
 'hers',
 'herself',
 'it',
 "it's",
 'its',
 'itself',
 'they',
 'them',
 'their',
 'theirs',
 'themselves',
 'what',
 'which',
 'who',
 'whom',
 'this',
 'that',
 "that'll",
 'these',
 'those',
 'am',
 'is',
 'are',
 'was',
 'were',
 'be',
 'been',
 'being',
 'have',
 'has',
 'had',
 'having',
```

```
             'do',
             'does',
             'did',
             'doing',
             'a',
             'an',
             'the',
             'and',
             'but',
             'if',
             'or',
             'because',
             'as',
             'until',
             'while',
             'of',
             'at',
             'by',
             'for',
             'with',
             'about',
             'against',
             'between',
             'into',
             'through',
             'during',
             'before',
             'after',
             'above',
             'below',
             'to',
             'from',
             'up',
             'down',
             'in',
             'out',
             'on',
             'off',
             'over',
             'under',
             'again',
             'further',
             'then',
             'once',
             'here',
             'there',
             'when',
             'where',
             'why',
             'how',
             'all',
             'any',
             'both',
             'each',
             'few',
             'more',
             'most',
             'other',
             'some',
             'such',
             'no',
             'nor',
             'not',
             'only',
             'own',
             'same',
```

```
 'so',
 'than',
 'too',
 'very',
 's',
 't',
 'can',
 'will',
 'just',
 'don',
 "don't",
 'should',
 "should've",
 'now',
 'd',
 'll',
 'm',
 'o',
 're',
 've',
 'y',
 'ain',
 'aren',
 "aren't",
 'couldn',
 "couldn't",
 'didn',
 "didn't",
 'doesn',
 "doesn't",
 'hadn',
 "hadn't",
 'hasn',
 "hasn't",
 'haven',
 "haven't",
 'isn',
 "isn't",
 'ma',
 'mightn',
 "mightn't",
 'mustn',
 "mustn't",
 'needn',
 "needn't",
 'shan',
 "shan't",
 'shouldn',
 "shouldn't",
 'wasn',
 "wasn't",
 'weren',
 "weren't",
 'won',
 "won't",
 'wouldn',
 "wouldn't"]
```

Count the number of stopwords

In [17]:
```
len(eng_stopwords)
```

Out[17]:
```
179
```

Exercise 2-1: Remove the stopwords from the dataset (similarly to how we removed punctuation above)

In [18]:
```
words_wo_puncts_stopwords = [i for i in words_wo_puncts if i not in eng_stopwords]
len(words_wo_puncts_stopwords)
```

Out[18]:
```
710578
```

Exercise 2-2: Find the number of unique words in the dataset now that the stop words have been removed

In [19]:
```
# unique words without stopwords
unique_words = set(words_wo_puncts_stopwords)
len(unique_words)
```

Out[19]:
```
39586
```

Exercise 2-3: Find the top 20 highest frequency words now that we have removed the stopwords

In [21]:
```
# top 20 highest freq words after removing stopwords
pd.Series(words_wo_puncts_stopwords).value_counts()[:20]
```

Out[21]:

	count
film	9517
one	5852
movie	5771
like	3690
even	2565
time	2411
good	2411
story	2169
would	2109
much	2049
character	2020
also	1967
get	1949
two	1911
well	1906
characters	1859
first	1836
--	1815
see	1749
way	1693

dtype: int64

Find the words that are used only once in the corpus (and print the first few).

In [22]:
```
# 20 words that are used only once in corpus using hapaxes() function
nltk.FreqDist(words_wo_puncts_stopwords).hapaxes()[:20]
```

Out[22]:
```
['looooot',
 'schnazzy',
 'timex',
 'indiglo',
 'jessalyn',
 'gilsig',
 'ruber',
 'jaleel',
 'balki',
 'wavers',
 'statistics',
 'snapshot',
 'guesswork',
 'maryam',
 'daylights',
 'terraformed',
 'stagnated',
 'napolean',
 'millimeter',
 'enmeshed']
```

Exercise 2-4: Use the PorterStemmer to stem the words in the dataset.

Display the first few words.

In [30]:
```
from nltk.stem import PorterStemmer

ps = PorterStemmer()
words_wo_puncts_stopwords_stemmed = [ps.stem(i) for i in words_wo_puncts_stopwords]

for i in range(20):
  print(words_wo_puncts_stopwords[i],": ",words_wo_puncts_stopwords_stemmed[i])
```

```
plot :  plot
two :  two
teen :  teen
couples :  coupl
go :  go
church :  church
party :  parti
drink :  drink
drive :  drive
get :  get
accident :  accid
one :  one
guys :  guy
dies :  die
girlfriend :  girlfriend
continues :  continu
see :  see
life :  life
nightmares :  nightmar
deal :  deal
```

Exercise 2-5: Use the WordNetLemmatizer to lemmatize the words in the dataset.

Display the first few words.

In [31]:
```python
from nltk import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

words_wo_puncts_stopwords_lemma = [lemmatizer.lemmatize(i) for i in words_wo_puncts_stop

for i in range(20):
  print(words_wo_puncts_stopwords[i],": ",words_wo_puncts_stopwords_lemma[i])
```

```
plot :  plot
two :  two
teen :  teen
couples :  couple
go :  go
church :  church
party :  party
drink :  drink
drive :  drive
get :  get
accident :  accident
one :  one
guys :  guy
dies :  dy
girlfriend :  girlfriend
continues :  continues
see :  see
life :  life
nightmares :  nightmare
deal :  deal
```

Exercise 2-6:

a) How many unique words are there once stemming is applied? (show the that performs the computation and outputs the result)

b) How many unique words are there once lemmatization is applied? (show the code that performs the computation and outputs the result)

In [32]:
```python
print("Unique words after stemming: ", len(set(words_wo_puncts_stopwords_stemmed)))
print("Unique words after lemmatization: ", len(set(words_wo_puncts_stopwords_lemma)))
```

```
Unique words after stemming:  26101
Unique words after lemmatization:  35172
```

Part 3. Tokenization

Exercise 3-1: Use the Penn Tree Bank tokenizer to tokenize the sentence below

Print the tokens that the tokenizer produces.

In [33]:
```python
from nltk.tokenize import TreebankWordTokenizer
s = 'Please pay $100.55 to settle your bill.  Send confirmation to confirm@gmail.com.'

tokenizer = TreebankWordTokenizer()
s_tokens = tokenizer.tokenize(s)
print(s_tokens)
```

```
['Please', 'pay', '$', '100.55', 'to', 'settle', 'your', 'bill.', 'Send', 'confirmation', 'to', 'confirm', '@', 'gmail.com', '.']
```

Part 4: Levenshtein Distance & Alignment

Relevant nltk documentation: https://www.nltk.org/api/nltk.metrics.distance.html

Exercise 4-1: Use the nltk functions edit_distance to compute the Levenshtein edit-distance between the strings "intention" and "execution"

In [35]:
```python
from nltk.metrics.distance import edit_distance

w1 = "intention"
w2 = "execution"
dist = edit_distance(w1,w2)
dist
```

Out[35]:
```
5
```

Exercise 4-2: Use the nltk function edit_distance_align to compute the minimum Levenshtein edit-distance based alignment mapping between the two strings "intention" and "execution"

In [38]:
```python
from nltk.metrics.distance import edit_distance_align

w1 = "intention"
w2 = "execution"
dist_align = edit_distance_align(w1,w2)
dist_align
```

Out[38]:
```
[(0, 0),
 (1, 1),
 (2, 2),
 (3, 3),
 (4, 4),
 (5, 5),
 (6, 6),
 (7, 7),
 (8, 8),
 (9, 9)]
```

In [37]:
```python
help(edit_distance_align)
```

```
Help on function edit_distance_align in module nltk.metrics.distance:

edit_distance_align(s1, s2, substitution_cost=1)
    Calculate the minimum Levenshtein edit-distance based alignment
    mapping between two strings. The alignment finds the mapping
    from string s1 to s2 that minimizes the edit distance cost.
    For example, mapping "rain" to "shine" would involve 2
    substitutions, 2 matches and an insertion resulting in
    the following mapping:
    [(0, 0), (1, 1), (2, 2), (3, 3), (4, 4), (4, 5)]
    NB: (0, 0) is the start state without any letters associated
    See more: https://web.stanford.edu/class/cs124/lec/med.pdf

    In case of multiple valid minimum-distance alignments, the
    backtrace has the following operation precedence:

    1. Substitute s1 and s2 characters
```

```
2. Skip s1 character
3. Skip s2 character

The backtrace is carried out in reverse string order.

This function does not support transposition.

:param s1, s2: The strings to be aligned
:type s1: str
:type s2: str
:type substitution_cost: int
:rtype: List[Tuple(int, int)]
```

In [ ]:

```

```