Activity 3: Normalization

Instructions:

- Please download the provided IPython Notebook (ipynb) file and open it in Google Colab. Once opened, enter your code in the same file directly beneath the relevant question's code block.
- Insert a text block below your code to briefly explain it, mentioning any libraries or functions utilized. Conclude your activity with a comprehensive explanation of your overall approach, aiming for about 200 words, in the final section of the notebook.

- Submit

1. The IPython Notebook (ipynb) file.
2. A PDF version of the notebook (converted from ipynb).

- The similarity score should be less than 15%

Text Preprocessing Beyond Tokenization

Word normalization

Word normalization is a text preprocessing technique used in natural language processing (NLP) to standardize and simplify words or tokens in a text document. The goal of word normalization is to make text data more consistent and manageable for analysis. This process can involve various transformations, such as converting all text to lowercase, removing punctuation, expanding contractions, and performing tasks like stemming or lemmatization to reduce words to their base or dictionary forms. Word normalization helps improve the accuracy and effectiveness of NLP tasks by reducing the complexity of text data and ensuring that similar words are treated as equivalent.

In [1]:
```
import re

# for using NLTK
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
from nltk.corpus import stopwords

# for using SpaCy
import spacy

# for HuggingFace
!pip install transformers
# !pip install ftfy
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.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...
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dis
t-packages (4.35.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-pa
ckages (from transformers) (3.13.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.16.4 in /usr/local/li
b/python3.10/dist-packages (from transformers) (0.20.3)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist
-packages (from transformers) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/
dist-packages (from transformers) (23.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist
-packages (from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.1
0/dist-packages (from transformers) (2023.12.25)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-pa
ckages (from transformers) (2.31.0)
Requirement already satisfied: tokenizers<0.19,>=0.14 in /usr/local/lib/pyth
on3.10/dist-packages (from transformers) (0.15.1)
Requirement already satisfied: safetensors>=0.3.1 in /usr/local/lib/python3.
10/dist-packages (from transformers) (0.4.2)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-
packages (from transformers) (4.66.1)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.1
0/dist-packages (from huggingface-hub<1.0,>=0.16.4->transformers) (2023.6.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/
python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->transformers) (
4.9.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/py
thon3.10/dist-packages (from requests->transformers) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dis
t-packages (from requests->transformers) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.
10/dist-packages (from requests->transformers) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.
10/dist-packages (from requests->transformers) (2024.2.2)
```

- Import/Download the required libraries/models

In [2]:
```
# trick to wrap text to the viewing window for this notebook
# Ref: https://stackoverflow.com/questions/58890109/line-wrapping-in-collaboratory-google-results
#helps improve the readability and formatting of code output in the notebook.
from IPython.display import HTML, display

def set_css():
  display(HTML('''
  <style>
    pre {
        white-space: pre-wrap;
    }
  </style>
  '''))
get_ipython().events.register('pre_run_cell', set_css)
```

Revisiting Tokenization :TreebankWordTokenizer

The **Treebank Word Tokenizer** is a text processing tool used in natural language processing (NLP) to split text into individual words or tokens. It follows the tokenization conventions and standards of the Penn Treebank corpus

In [3]:
```
from nltk.tokenize import TreebankWordTokenizer
text="Hello everyone. Welcome to NLP Course."
tokenizer = TreebankWordTokenizer()
tokenizer.tokenize(text)
```

Out[3]:
```
['Hello', 'everyone.', 'Welcome', 'to', 'NLP', 'Course', '.']
```

- TreebankWordTokenizer splits the sentence into words called as tokens)

Using Regular Expression

RegexpTokenizer is a text processing tool provided by the Natural Language Toolkit (NLTK) library in Python. It is used to tokenize (split) text into individual tokens (words or phrases) based on a specified regular expression pattern.

In [4]:
```
from nltk.tokenize import RegexpTokenizer

tokenizer = RegexpTokenizer("[\w']+")#[\w']+ as a whole matches sequences
of word characters (letters, digits, underscores) and single quotes in a
string
text = "Let's see how it's working. We also have digits like 123 and 010"
tokenizer.tokenize(text)
```

Out[4]:
```
["Let's",
 'see',
 'how',
 "it's",
 'working',
 'We',
 'also',
 'have',
 'digits',
 'like',
 '123',
 'and',
 '010']
```

Question 1: Copy the above snippet code and Modify the above regular expression to match only **digits** from the above given text.

In [5]:
```
#CODE HERE
from nltk.tokenize import RegexpTokenizer

tokenizer = RegexpTokenizer("[\d]+")#[\w']+ as a whole matches sequences of
word characters (letters, digits, underscores) and single quotes in a
string
text = "Let's see how it's working. We also have digits like 123 and 010"
tokenizer.tokenize(text)
```

Out[5]:
```
['123', '010']
```

- In this modified regular expression '\d' matches any digit and '+' means digit occurs one or more times.

(Tutorial) Stemming and Lemmatization using NLTK

Stemming is a text normalization technique in natural language processing and information retrieval. It involves reducing words to their root or base form, often by removing suffixes or prefixes. The goal of stemming is to convert words with the same meaning but different forms into a common base form so that they can be treated as equivalent during text analysis and retrieval. Stemming helps improve information retrieval and text processing tasks by reducing the complexity of words while maintaining their core meaning. Common stemming algorithms include the Porter Stemmer and Snowball Stemmer.

Porter Stemmer

The **Porter Stemmer** is a well-known algorithm for stemming in natural language processing. It was designed to reduce words to their root or base form by removing common suffixes. Stemming is the process of reducing words to their linguistic root or base form to simplify text analysis and improve information retrieval.For example, it can convert words like "running," "runs," and "ran" to their common root "run."

Let's see how we can perform stemming and lemmatization using NLTK library...

In [6]:
```
# importing PorterStemmer class from nltk.stem module
from nltk.stem import PorterStemmer
porter = PorterStemmer()    # instantiating an object of the PorterStemmer
class

stem = porter.stem('cats')    # calling the stemmer algorithm on the
desired word
print(f"'cats' after stemming: {stem}")

stem = porter.stem('better')
print(f"'better' after stemming: {stem}")

stem = porter.stem('abaci')
print(f"'abaci' after stemming: {stem}")

stem = porter.stem('aardwolves')
print(f"'aardwolves' after stemming: {stem}")

stem = porter.stem('generically')
print(f"'generically' after stemming: {stem}")
```

```
'cats' after stemming: cat
'better' after stemming: better
'abaci' after stemming: abaci
'aardwolves' after stemming: aardwolv
'generically' after stemming: gener
```

Lemmatization

Lemmatization is a natural language processing technique that reduces words to their base or dictionary form, known as a "lemma." Unlike stemming, which often involves removing suffixes to approximate a word's root, lemmatization considers the word's context and grammatical meaning. The goal is to transform different inflected forms of a word into a common base form. Lemmatization is particularly useful for maintaining the grammatical correctness of words in text analysis and information retrieval tasks.

WordNet Lemmatizer

The **WordNet Lemmatizer** is a lemmatization tool based on WordNet, a lexical database of the English language. WordNet groups words into sets of synonyms called "synsets" and provides a rich lexical and semantic structure for the English language. The WordNet Lemmatizer uses this semantic information to perform lemmatization, which is the process of reducing words to their base or dictionary form (lemma)

In [7]:
```
import nltk
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[7]:
```
True
```

- Download the WordNet from NLTK.

In [8]:
```
# importing WordNet-based lemmatizer class from nltk.stem module
from nltk.stem import WordNetLemmatizer
import nltk
nltk.download('omw-1.4')

lemmatizer = WordNetLemmatizer()    # instantiating an object of the
WordNetLemmatizer class

lemma = lemmatizer.lemmatize('cats')    # calling the lemmatization
algorithm on the desired word
print(f"'cats' after lemmatization: {lemma}")

lemma = lemmatizer.lemmatize('better')
print(f"'better' after lemmatization: {lemma}")

lemma = lemmatizer.lemmatize('abaci')
print(f"'abaci' after lemmatization: {lemma}")

lemma = lemmatizer.lemmatize('aardwolves')
print(f"'aardwolves' after lemmatization: {lemma}")

lemma = lemmatizer.lemmatize('generically')
print(f"'generically' after lemmatization: {lemma}")

print("\n\n\n")
lemma = lemmatizer.lemmatize('better', pos='a')   # 'a' denoted ADJECTIVE
part-of-speech
print(f"'better' (as an adjective) after lemmatization: {lemma}")
```

```
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
'cats' after lemmatization: cat
'better' after lemmatization: better
'abaci' after lemmatization: abacus
'aardwolves' after lemmatization: aardwolf
'generically' after lemmatization: generically



'better' (as an adjective) after lemmatization: good
```

Stemming on text string

In [9]:
```
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer

def stem_text(text):
    # Initialize the Porter Stemmer
    stemmer = PorterStemmer()

    # Tokenize the text into words
    words = word_tokenize(text)

    # Apply the stemmer to each word and join them back into a text
    stemmed_text = ' '.join([stemmer.stem(word) for word in words])

    return stemmed_text

# Example usage:
text = "He is jumping, and he jumped over the jumps."
stemmed_text = stem_text(text)
print(stemmed_text)
```

```
he is jump , and he jump over the jump .
```

In [10]:
```
# This is the text on which you have to perform stemming; taken from
Wikipedia.
text = "In linguistic morphology and information retrieval, stemming is the
process of reducing inflected (or sometimes derived) words to their word
stem, base or root form; generally a written word form. The stem need not
be identical to the morphological root of the word; it is usually
sufficient that related words map to the same stem, even if this stem is
not in itself a valid root."
print("Given text:")
print(text)
```

```
Given text:
In linguistic morphology and information retrieval, stemming is the process
of reducing inflected (or sometimes derived) words to their word stem, base
or root form; generally a written word form. The stem need not be identical
to the morphological root of the word; it is usually sufficient that related
words map to the same stem, even if this stem is not in itself a valid root.
```

Tutorial 2

In [11]:
```
#CODE BLOCK 1
en_stopwords = set(stopwords.words('english'))
def remove_punc(text_string):
  return re.sub('[^a-zA-Z0-9 ]', '', text_string.lower())

def remove_stopwords(text_string):
  return [ token for token in text_string.split(' ') if token not in
en_stopwords ]

# applying punctuation removal to the text
unpunc_text = remove_punc(text)
print("After punctuation removal:")
print(unpunc_text)

# # applying stopword removal to the text
clean_text = remove_stopwords(unpunc_text)
print("\n\nAfter stopword removal:")
print(clean_text)
```

```
After punctuation removal:
in linguistic morphology and information retrieval stemming is the process o
f reducing inflected or sometimes derived words to their word stem base or r
oot form generally a written word form the stem need not be identical to the
morphological root of the word it is usually sufficient that related words m
ap to the same stem even if this stem is not in itself a valid root


After stopword removal:
['linguistic', 'morphology', 'information', 'retrieval', 'stemming', 'proces
s', 'reducing', 'inflected', 'sometimes', 'derived', 'words', 'word', 'ste
m', 'base', 'root', 'form', 'generally', 'written', 'word', 'form', 'stem',
'need', 'identical', 'morphological', 'root', 'word', 'usually', 'sufficien
t', 'related', 'words', 'map', 'stem', 'even', 'stem', 'valid', 'root']
```

- [^a-zA-Z0-9]: The means other than characters other than lowercase, uppercase, digits, and space -The above code removed all the punctuations and stop words.

Question 2. Perform stemming on the cleaned text(from tutorial2-code block 1) above using the Porter Stemmer from NLTK.

Hint: import PorterStemmer from nltk.stem

In [12]:
```
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer

def stem_text(text):
    # Initialize the Porter Stemmer
    stemmer = PorterStemmer()

    # Tokenize the text into words
    words = word_tokenize(text)

    # Apply the stemmer to each word and join them back into a text
    stemmed_text = ' '.join([stemmer.stem(word) for word in words])

    return stemmed_text

# Example usage:
print("Before Stemming: \n"+' '.join(clean_text))
print()
stemmed_text = stem_text(' '.join(clean_text))
print("After Stemming: \n"+stemmed_text)
```

```
Before Stemming:
linguistic morphology information retrieval stemming process reducing inflec
ted sometimes derived words word stem base root form generally written word
form stem need identical morphological root word usually sufficient related
words map stem even stem valid root

After Stemming:
linguist morpholog inform retriev stem process reduc inflect sometim deriv w
ord word stem base root form gener written word form stem need ident morphol
og root word usual suffici relat word map stem even stem valid root
```

- The above code applies stemming on the clean_text generated from tutorial-2.
- For stemming, the code uses PorterStemmer.

Lemmatization on text string

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

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

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

In [14]:
```
from nltk.stem import WordNetLemmatizer

# Initialize the WordNet Lemmatizer
lemmatizer = WordNetLemmatizer()

# Example text to lemmatize
text = "There are like more than 100 foxes and lions in this forest."

# Tokenize the text into words
words = text.split()

# Lemmatize each word and join them back into a sentence
lemmatized_text = ' '.join([lemmatizer.lemmatize(word) for word in words])

# Print the original and lemmatized text
print("Original Text:", text)
print("Lemmatized Text:", lemmatized_text)
```

```
Original Text: There are like more than 100 foxes and lions in this forest.
Lemmatized Text: There are like more than 100 fox and lion in this forest.
```

Question 3. Perform lemmatization on the same cleaned text(from tutorial2-code block 1) above using NLTK's lemmatizer.

Hint:import WordNetLemmatizer from nltk.stem

In [15]:
```
# apply NLTK's lemmatizer on the cleaned text (after punctuation and
stopwords are removed) below this comment
#CODE HERE
from nltk.stem import WordNetLemmatizer

# Initialize the WordNet Lemmatizer
lemmatizer = WordNetLemmatizer()

# # Example text to lemmatize
# text = "There are like more than 100 foxes and lions in this forest."

# # Tokenize the text into words
# words = text.split()

# Lemmatize each word and join them back into a sentence
lemmatized_text = ' '.join([lemmatizer.lemmatize(word) for word in
clean_text])

# Print the original and lemmatized text
print("Original Text:", ' '.join(clean_text))
print("Lemmatized Text:", lemmatized_text)
```

```
Original Text: linguistic morphology information retrieval stemming process
reducing inflected sometimes derived words word stem base root form generall
y written word form stem need identical morphological root word usually suff
icient related words map stem even stem valid root
Lemmatized Text: linguistic morphology information retrieval stemming proces
s reducing inflected sometimes derived word word stem base root form general
ly written word form stem need identical morphological root word usually suf
ficient related word map stem even stem valid root
```

- The above code usesWordNetLemmatizer for lemmatization.
- It lemmatizes the clean_text.
- Since the clean_text is already split in words we do not need to split it again.

(Tutorial) Subword Tokenization using HuggingFace

Hugging Face is used for subword tokenization by offering NLP practitioners access to pre-trained subword tokenizers and models. Hugging Face's "transformers" library offers pre-trained models and tokenizers, such as Byte Pair Encoding (BPE) and SentencePiece, which are widely used for subword tokenization

Subword tokenization is a text processing technique used in natural language processing (NLP) to break down words into smaller units, often subword pieces. This approach is particularly useful for handling languages with complex morphology or when dealing with out-of-vocabulary words. Subword tokenization methods like Byte-Pair Encoding (BPE) and SentencePiece divide text into subword units, such as character-level tokens or subword pieces, allowing NLP models to work with a more extensive and adaptable vocabulary. This technique improves the handling of rare words and enhances the performance of NLP models on a wide range of languages and tasks

In [16]:
```
!pip install tokenizers
#This is a JSON file that contains the vocabulary (i.e., the set of words
and subword pieces) used by the GPT-2 model
!wget https://s3.amazonaws.com/models.huggingface.co/bert/gpt2-medium-
vocab.json
!wget https://s3.amazonaws.com/models.huggingface.co/bert/gpt2-merges.txt
```

```
Requirement already satisfied: tokenizers in /usr/local/lib/python3.10/dist-
packages (0.15.1)
Requirement already satisfied: huggingface_hub<1.0,>=0.16.4 in /usr/local/li
b/python3.10/dist-packages (from tokenizers) (0.20.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-pa
```

```
ckages (from huggingface_hub<1.0,>=0.16.4->tokenizers) (3.13.1)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.1
0/dist-packages (from huggingface_hub<1.0,>=0.16.4->tokenizers) (2023.6.0)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-pa
ckages (from huggingface_hub<1.0,>=0.16.4->tokenizers) (2.31.0)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.10/dis
t-packages (from huggingface_hub<1.0,>=0.16.4->tokenizers) (4.66.1)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist
-packages (from huggingface_hub<1.0,>=0.16.4->tokenizers) (6.0.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/
python3.10/dist-packages (from huggingface_hub<1.0,>=0.16.4->tokenizers) (4.
9.0)
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.10/
dist-packages (from huggingface_hub<1.0,>=0.16.4->tokenizers) (23.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/py
thon3.10/dist-packages (from requests->huggingface_hub<1.0,>=0.16.4->tokeniz
ers) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dis
t-packages (from requests->huggingface_hub<1.0,>=0.16.4->tokenizers) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.
10/dist-packages (from requests->huggingface_hub<1.0,>=0.16.4->tokenizers) (
2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.
10/dist-packages (from requests->huggingface_hub<1.0,>=0.16.4->tokenizers) (
2024.2.2)
--2024-02-09 22:31:32--  https://s3.amazonaws.com/models.huggingface.co/ber
t/gpt2-medium-vocab.json
Resolving s3.amazonaws.com (s3.amazonaws.com)... 52.217.108.214, 54.231.128.
72, 16.182.32.8, ...
Connecting to s3.amazonaws.com (s3.amazonaws.com)|52.217.108.214|:443... con
nected.
HTTP request sent, awaiting response... 200 OK
Length: 1042301 (1018K) [application/json]
Saving to: ‘gpt2-medium-vocab.json’

gpt2-medium-vocab.j 100%[===================>]   1018K  --.-KB/s    in 0.04s

2024-02-09 22:31:32 (25.2 MB/s) - ‘gpt2-medium-vocab.json’ saved [1042301/10
42301]

--2024-02-09 22:31:32--  https://s3.amazonaws.com/models.huggingface.co/ber
t/gpt2-merges.txt
Resolving s3.amazonaws.com (s3.amazonaws.com)... 52.217.108.214, 54.231.128.
72, 16.182.32.8, ...
Connecting to s3.amazonaws.com (s3.amazonaws.com)|52.217.108.214|:443... con
nected.
HTTP request sent, awaiting response... 200 OK
Length: 456318 (446K) [text/plain]
Saving to: ‘gpt2-merges.txt’

gpt2-merges.txt     100%[===================>] 445.62K  --.-KB/s    in 0.02s

2024-02-09 22:31:32 (24.5 MB/s) - ‘gpt2-merges.txt’ saved [456318/456318]
```

Byte Pair Encoding (BPE) is a subword tokenization technique used in natural language processing (NLP) and text processing. It involves dividing text into subword units, typically based on frequency, to create a more flexible and adaptive vocabulary for language models.

In [17]:
```
from tokenizers import ByteLevelBPETokenizer
gpt2vocab = "gpt2-medium-vocab.json"
gpt2merges = "gpt2-merges.txt"

bpe = ByteLevelBPETokenizer(gpt2vocab, gpt2merges)
bpe_encoding = bpe.encode("The custom of delivering an address on
Inauguration Day started with the very first Inauguration—George
Washington's—on April 30, 1789.")
print(bpe_encoding.ids)
print(bpe_encoding.tokens)
```

```
[464, 2183, 286, 13630, 281, 2209, 319, 554, 7493, 3924, 3596, 2067, 351, 26
2, 845, 717, 554, 7493, 3924, 960, 20191, 2669, 447, 247, 82, 960, 261, 303
5, 1542, 11, 1596, 4531, 13]
['The', 'Ġcustom', 'Ġof', 'Ġdelivering', 'Ġan', 'Ġaddress', 'Ġon', 'ĠIn', 'a
ug', 'uration', 'ĠDay', 'Ġstarted', 'Ġwith', 'Ġthe', 'Ġvery', 'Ġfirst', 'ĠI
n', 'aug', 'uration', 'âĢĶ', 'George', 'ĠWashington', 'âĢ', 'Ļ', 's', 'âĢĶ',
'on', 'ĠApril', 'Ġ30', ',', 'Ġ17', '89', '.']
```

- Ġ represents space (' ').

Question 4: Collect the encoding ids which you generate from above snippet and now decode the ids to get back the given text string?

Hint: use decode() method

In [18]:
```
#CODE HERE
print("Encoded tokens:")
print(bpe_encoding.ids)
bpe_decoding = bpe.decode(bpe_encoding.ids)
print()
print("Decoded tokens:")
print(bpe_decoding)
```

```
Encoded tokens:
[464, 2183, 286, 13630, 281, 2209, 319, 554, 7493, 3924, 3596, 2067, 351, 26
2, 845, 717, 554, 7493, 3924, 960, 20191, 2669, 447, 247, 82, 960, 261, 303
5, 1542, 11, 1596, 4531, 13]

Decoded tokens:
The custom of delivering an address on Inauguration Day started with the ver
y first Inauguration—George Washington’s—on April 30, 1789.
```

- decode(): Used to get back the original token. Decodes the id's to generate the original text/tokens.

Explain briefly the changes you have made in the given Tasks in Today's activity (approximately 200 words)

-->

Question-1:

- I modified the regular expression such that it can accept only digits.
- I used '\d' which means any digit and '+' means the digit occurs one or more times.

Question-2:

- I have taken the cleaned data from tutorial-2 code block-1 and applied stemming on it.
- For Steeming there are many libraries but we have used Porter Stemmer which is from the NLTK package.
- Stemming is nothing but reducing the terms to stem, chopping off affixes crudely.
- In other words using fixed rules such as removing able, ing, etc. to derive the base word is called stemming.
- Stemming can be done using NLTK but not using Spacy, because Spacy does not support Stemming, it only supports Lemmatization.

Question 3:

- For the data that we performed stemming, not we will perform lemmatization.
- We used WordNetLemmatizer from NLTK for lemmatization. It lemmatizes word by word by looping the array.
- The clean_text is already split, so we do not need to split it again so I commented those lines.

Question 4:

- For question 4, we decoded the tokens to get back the original data.
- We used the encode() method to encode the sentence, which gave a unique id's for every token.
- And if only id's are given we can use the decode() method to get the original tokens/data/sentence.