Charis Fernandes, 26

St. Francis Institute of Technology, Mumbai-400 103
Department Of Information Technology

A.Y. 2022-2023
Class: BE-ITA/B, Semester: VII
Subject: Data Science Lab

Experiment – 4

1. **Aim:** To implement a Cognitive Computing Application

2. **Objectives:** Students should be able to design a solution for problem using Cognitive Computing.

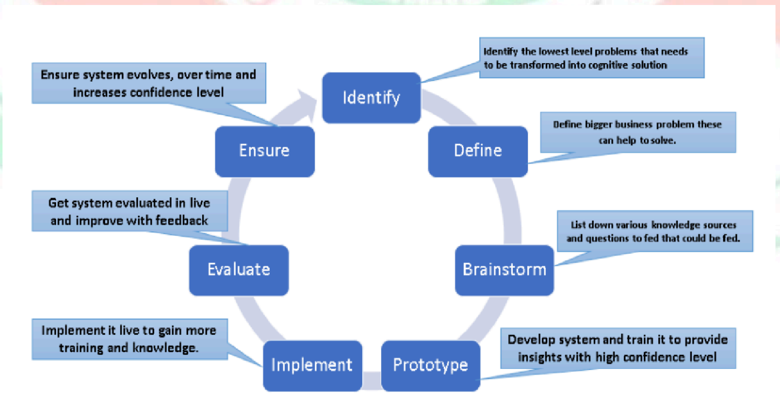
3. **Prerequisite:** Python basics

4. **Requirements:** PC, Python 3.9, Windows 10/ MacOS/ Linux, IDLE IDE

5. Pre-Experiment Exercise:

Theory:
- The Cognitive is the mental action to learning and acquiring through thought, experience and the senses.
- Cognitive computing is computerized model that simulates human thought process in complex situations where the answer may be ambiguous and uncertain.
- Cognitive computing systems can recognize, understand, analyze, memorize and take out best possible result as or near about the human brain.
- The basic idea behind this type of computing is that to develop the computer system(include hardware and software) who interacts with human like humans.
- To accomplish this, cognitive computing makes use of AI and underlying technologies.
- If you look at cognitive computing as an analog to the human brain, you need to analyze in context all types of data, from structured data in databases to unstructured data in text, images, voice, sensors, and video.

Design Principles of Cognitive Computing:


Identify
Identify the lowest level problems that needs to be transformed into cognitive solution
Define
Define bigger business problem these can help to solve.
Brainstorm
List down various knowledge sources and questions to fed that could be fed.
Prototype
Develop system and train it to provide insights with high confidence level
Implement
Implement it live to gain more training and knowledge.
Evaluate
Get system evaluated in live and improve with feedback
Ensure
Ensure system evolves, over time and increases confidence level

Phases in NLP:

Phonological Analysis:
- It is applied if input is speech.

Morphological Analysis
- Deals with understanding distinct words according to their morphemes.
- Eg: Unhappiness: broken down into three morphemes (prefix, stem, suffix).
- Stem is considered as free morpheme and prefix and suffix are considered are bound morphemes.

Lexical Analysis:
- Lexicon of a language means the collection of words and phrases in the language.
- Lexical analysis is dividing the whole chunk of text into paragraphs, sentences and words.
- Lexicon normalization is often needed in Lexical analysis.
- The most common lexicon normalization are:
  - Stemming: it is a rudimentary rule based process of stripping the suffixes. From word.
  - Lemmatization: organized procedure of obtaining the root form of the word by using dictionary and morphological analysis.

Syntactic Analysis:
- Deals with analyzing the words of a sentence so as to uncover the grammatical structure of the sentence.
- Eg: “Colorless green idea”
- Checked for dependency grammar and parts of speech tags .

Semantic Analysis:
- Determines possible meaning of the sentence by focusing on the interactions among word level meanings in the sentence.

Discourse Integration:
- Focuses on the properties of the text as a whole that convey meaning by making connections between component sentences.

Pragmatic Analysis:
- Explains how extra meaning is read into texts without actually being encoded in them.
- It helps user to discover intended effect by applying set of rules that characterize cooperative dialogues.

6. Laboratory Exercise

A. Procedure

i. Use google colab for programming.
ii. Import nltk package.
iii. Demonstrate all phases of NLP on a given text.
iv. Add relevant comments in your programs and execute the code. Test it for various cases.

7. Post-Experiments Exercise:

A. Extended Theory:

a. Explain design Principles of Cognitive Computing.

B. Post Lab Program:

a. Select a application of your choice in domain like health care, banking, finance and implement

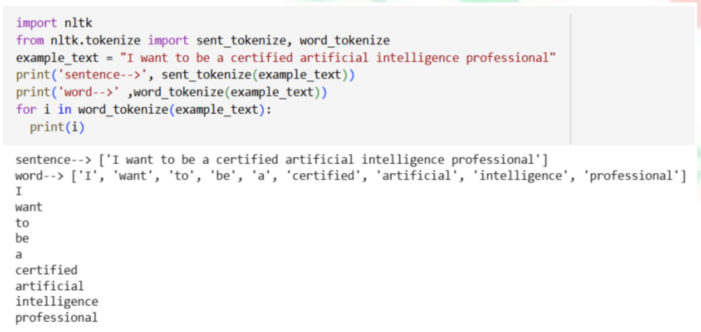
C. Conclusion:

1. Write what was performed in the program (s) .
2. What is the significance of program and what Objective is achieved?

D. References:

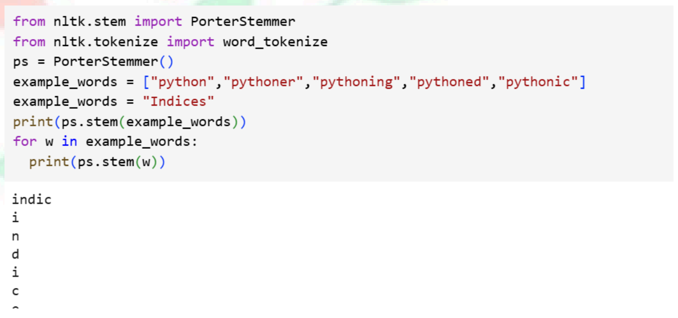
[1]Judith S. Hurwitz, Marcia Kaufman, Adrian Bowles, “Cognitive Computing and Big Data Analytics”, Wiley India, 2015.

Q. Demonstrate all phases of NLP on a given text.



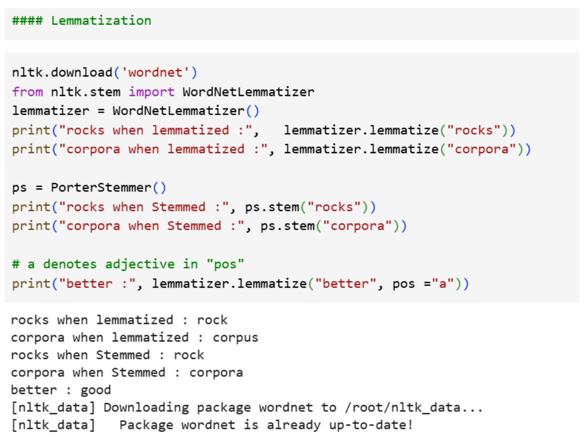
```
import nltk
from nltk.tokenize import sent_tokenize, word_tokenize
example_text = "I want to be a certified artificial intelligence professional"
print('sentence-->', sent_tokenize(example_text))
print('word-->' ,word_tokenize(example_text))
for i in word_tokenize(example_text):
  print(i)
```

```
sentence--> ['I want to be a certified artificial intelligence professional']
word--> ['I', 'want', 'to', 'be', 'a', 'certified', 'artificial', 'intelligence', 'professional']
I
want
to
be
a
certified
artificial
intelligence
professional
```



```
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
ps = PorterStemmer()
example_words = ["python","pythoner","pythoning","pythoned","pythonic"]
example_words = "Indices"
print(ps.stem(example_words))
for w in example_words:
  print(ps.stem(w))
```

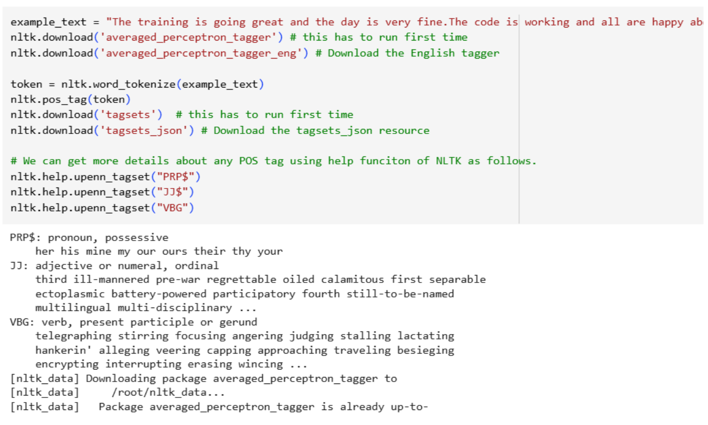
```
indic
i
n
d
i
c
```



```
#### Lemmatization
```

```python
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
print("rocks when lemmatized :",   lemmatizer.lemmatize("rocks"))
print("corpora when lemmatized :", lemmatizer.lemmatize("corpora"))

ps = PorterStemmer()
print("rocks when Stemmed :", ps.stem("rocks"))
print("corpora when Stemmed :", ps.stem("corpora"))

# a denotes adjective in "pos"
print("better :", lemmatizer.lemmatize("better", pos ="a"))
```

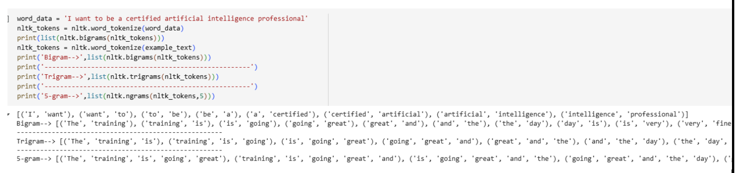
```
rocks when lemmatized : rock
corpora when lemmatized : corpus
rocks when Stemmed : rock
corpora when Stemmed : corpora
better : good
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```



```python
example_text = "The training is going great and the day is very fine.The code is working and all are happy ab
nltk.download('averaged_perceptron_tagger') # this has to run first time
nltk.download('averaged_perceptron_tagger_eng') # Download the English tagger

token = nltk.word_tokenize(example_text)
nltk.pos_tag(token)
nltk.download('tagsets')  # this has to run first time
nltk.download('tagsets_json') # Download the tagsets_json resource

# We can get more details about any POS tag using help funciton of NLTK as follows.
nltk.help.upenn_tagset("PRP$")
nltk.help.upenn_tagset("JJ$")
nltk.help.upenn_tagset("VBG")
```

```
PRP$: pronoun, possessive
    her his mine my our ours their thy your
JJ: adjective or numeral, ordinal
    third ill-mannered pre-war regrettable oiled calamitous first separable
    ectoplasmic battery-powered participatory fourth still-to-be-named
    multilingual multi-disciplinary ...
VBG: verb, present participle or gerund
    telegraphing stirring focusing angering judging stalling lactating
    hankerin' alleging veering capping approaching traveling besieging
    encrypting interrupting erasing wincing ...
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
```

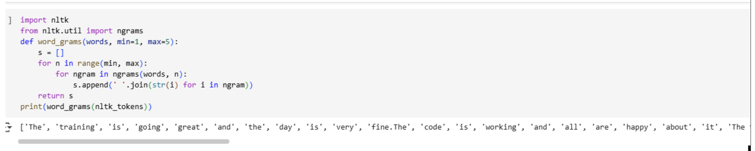
bigrams/trigrams/ngrams



```python
word_data = 'I want to be a certified artificial intelligence professional'
nltk_tokens = nltk.word_tokenize(word_data)
print(list(nltk.bigrams(nltk_tokens)))
nltk_tokens = nltk.word_tokenize(example_text)
print('Bigram-->',list(nltk.bigrams(nltk_tokens)))
print('-------------------------------------------------')
print('Trigram-->',list(nltk.trigrams(nltk_tokens)))
print('-------------------------------------------------')
print('5-gram-->',list(nltk.ngrams(nltk_tokens,5)))
```

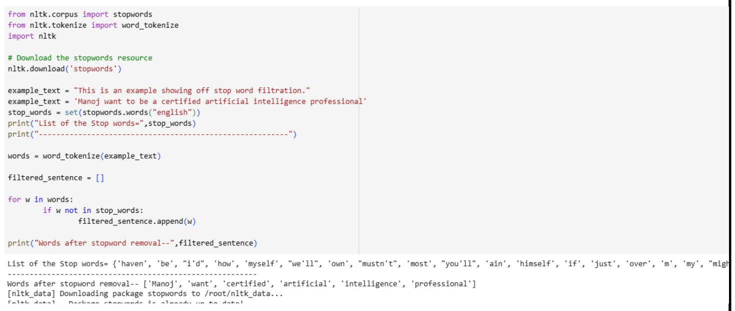
```
[('I', 'want'), ('want', 'to'), ('to', 'be'), ('be', 'a'), ('a', 'certified'), ('certified', 'artificial'), ('artificial', 'intelligence'), ('intelligence', 'professional')]
Bigram--> [('The', 'training'), ('training', 'is'), ('is', 'going'), ('going', 'great'), ('great', 'and'), ('and', 'the'), ('the', 'day'), ('day', 'is'), ('is', 'very'), ('very', 'fine
-------------------------------------------------
Trigram--> [('The', 'training', 'is'), ('training', 'is', 'going'), ('is', 'going', 'great'), ('going', 'great', 'and'), ('great', 'and', 'the'), ('and', 'the', 'day'), ('the', 'day',
-------------------------------------------------
5-gram--> [('The', 'training', 'is', 'going', 'great'), ('training', 'is', 'going', 'great', 'and'), ('is', 'going', 'great', 'and', 'the'), ('going', 'great', 'and', 'the', 'day'), ('
```

Printing all combinations of n-grams



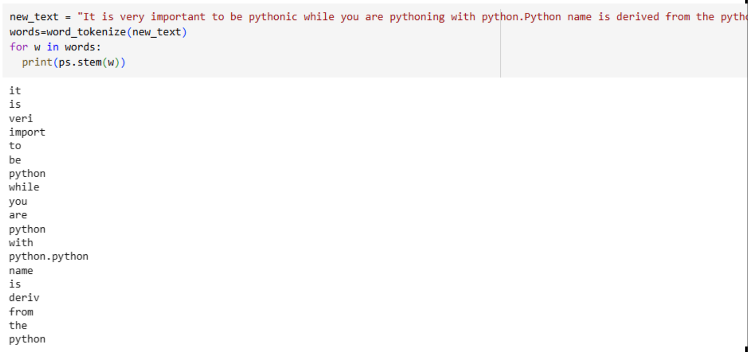
```python
import nltk
from nltk.util import ngrams
def word_grams(words, min=1, max=5):
    s = []
    for n in range(min, max):
        for ngram in ngrams(words, n):
            s.append(' '.join(str(i) for i in ngram))
    return s
print(word_grams(nltk_tokens))
```

```
['The', 'training', 'is', 'going', 'great', 'and', 'the', 'day', 'is', 'very', 'fine.The', 'code', 'is', 'working', 'and', 'all', 'are', 'happy', 'about', 'it', 'The
```



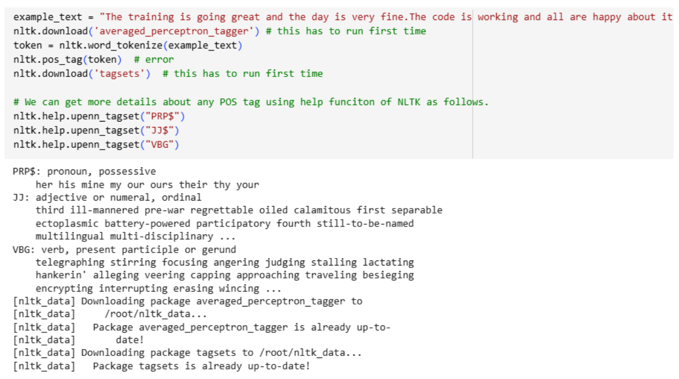
```python
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk

# Download the stopwords resource
nltk.download('stopwords')

example_text = "This is an example showing off stop word filtration."
example_text = 'Manoj want to be a certified artificial intelligence professional'
stop_words = set(stopwords.words("english"))
print("List of the Stop words=",stop_words)
print("--------------------------------------------------------")

words = word_tokenize(example_text)

filtered_sentence = []

for w in words:
        if w not in stop_words:
                filtered_sentence.append(w)

print("Words after stopword removal--",filtered_sentence)
```

```
List of the Stop words= {'haven', 'be', "i'd", 'how', 'myself', "we'll", 'own', "mustn't", 'most', "you'll", 'ain', 'himself', 'if', 'just', 'over', 'm', 'my', "migh
--------------------------------------------------------
Words after stopword removal-- ['Manoj', 'want', 'certified', 'artificial', 'intelligence', 'professional']
[nltk_data] Downloading package stopwords to /root/nltk_data...
```



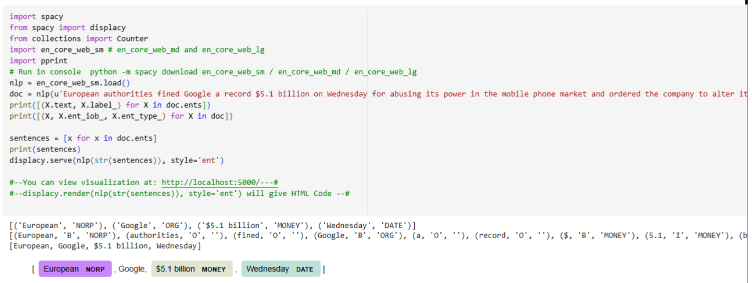
```python
new_text = "It is very important to be pythonic while you are pythoning with python.Python name is derived from the pytho
words=word_tokenize(new_text)
for w in words:
  print(ps.stem(w))
```

```
it
is
veri
import
to
be
python
while
you
are
python
with
python.python
name
is
deriv
from
the
python
```



```python
example_text = "The training is going great and the day is very fine.The code is working and all are happy about it
nltk.download('averaged_perceptron_tagger') # this has to run first time
token = nltk.word_tokenize(example_text)
nltk.pos_tag(token)  # error
nltk.download('tagsets')  # this has to run first time

# We can get more details about any POS tag using help funciton of NLTK as follows.
nltk.help.upenn_tagset("PRP$")
nltk.help.upenn_tagset("JJ$")
nltk.help.upenn_tagset("VBG")
```

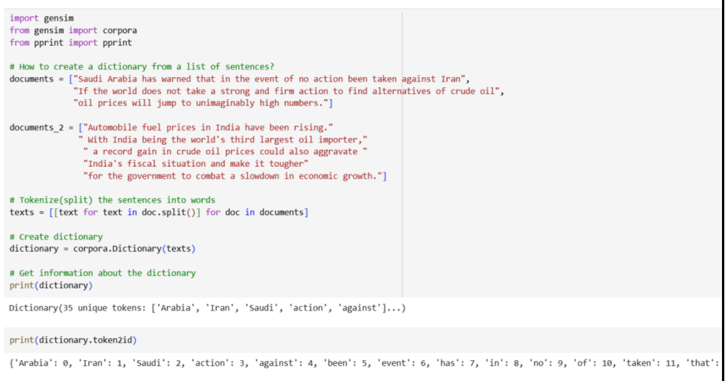
```
PRP$: pronoun, possessive
    her his mine my our ours their thy your
JJ: adjective or numeral, ordinal
    third ill-mannered pre-war regrettable oiled calamitous first separable
    ectoplasmic battery-powered participatory fourth still-to-be-named
    multilingual multi-disciplinary ...
VBG: verb, present participle or gerund
    telegraphing stirring focusing angering judging stalling lactating
    hankerin' alleging veering capping approaching traveling besieging
    encrypting interrupting erasing wincing ...
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package tagsets to /root/nltk_data...
[nltk_data]   Package tagsets is already up-to-date!
```



```python
import spacy
from spacy import displacy
from collections import Counter
import en_core_web_sm # en_core_web_md and en_core_web_lg
import pprint
# Run in console  python -m spacy download en_core_web_sm / en_core_web_md / en_core_web_lg
nlp = en_core_web_sm.load()
doc = nlp(u'European authorities fined Google a record $5.1 billion on Wednesday for abusing its power in the mobile phone market and ordered the company to alter it
print([(X.text, X.label_) for X in doc.ents])
print([(X, X.ent_iob_, X.ent_type_) for X in doc])

sentences = [x for x in doc.ents]
print(sentences)
displacy.serve(nlp(str(sentences)), style='ent')

#--You can view visualization at: http://localhost:5000/---#
#--displacy.render(nlp(str(sentences)), style='ent') will give HTML Code --#
```

```
[('European', 'NORP'), ('Google', 'ORG'), ('$5.1 billion', 'MONEY'), ('Wednesday', 'DATE')]
[(European, 'B', 'NORP'), (authorities, 'O', ''), (fined, 'O', ''), (Google, 'B', 'ORG'), (a, 'O', ''), (record, 'O', ''), ($, 'B', 'MONEY'), (5.1, 'I', 'MONEY'), (b
[European, Google, $5.1 billion, Wednesday]
```

[European NORP , Google, $5.1 billion MONEY , Wednesday DATE]

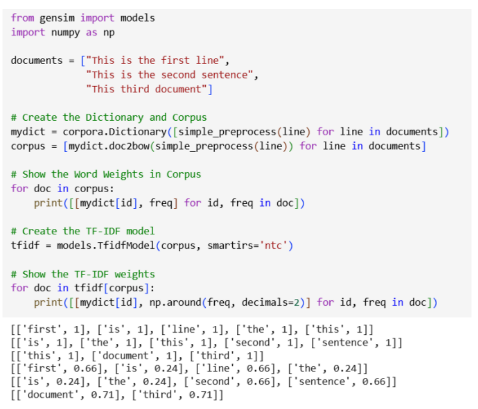


```python
import gensim
from gensim import corpora
from pprint import pprint

# How to create a dictionary from a list of sentences?
documents = ["Saudi Arabia has warned that in the event of no action been taken against Iran",
             "If the world does not take a strong and firm action to find alternatives of crude oil",
             "oil prices will jump to unimaginably high numbers."]

documents_2 = ["Automobile fuel prices in India have been rising."
               " With India being the world's third largest oil importer,"
                " a record gain in crude oil prices could also aggravate "
                "India's fiscal situation and make it tougher"
                "for the government to combat a slowdown in economic growth."]

# Tokenize(split) the sentences into words
texts = [[text for text in doc.split()] for doc in documents]

# Create dictionary
dictionary = corpora.Dictionary(texts)

# Get information about the dictionary
print(dictionary)
```

```
Dictionary(35 unique tokens: ['Arabia', 'Iran', 'Saudi', 'action', 'against']...)
```

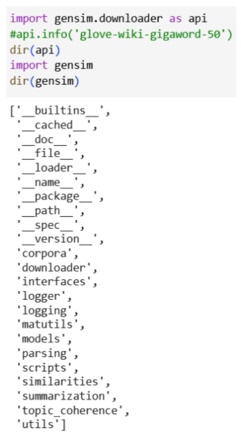
```python
print(dictionary.token2id)
```

```
{'Arabia': 0, 'Iran': 1, 'Saudi': 2, 'action': 3, 'against': 4, 'been': 5, 'event': 6, 'has': 7, 'in': 8, 'no': 9, 'of': 10, 'taken': 11, 'that':
```



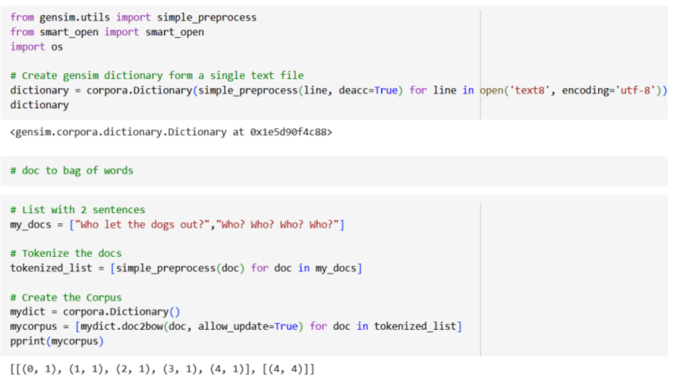
```python
from gensim import models
import numpy as np

documents = ["This is the first line",
             "This is the second sentence",
             "This third document"]

# Create the Dictionary and Corpus
mydict = corpora.Dictionary([simple_preprocess(line) for line in documents])
corpus = [mydict.doc2bow(simple_preprocess(line)) for line in documents]

# Show the Word Weights in Corpus
for doc in corpus:
    print([[mydict[id], freq] for id, freq in doc])

# Create the TF-IDF model
tfidf = models.TfidfModel(corpus, smartirs='ntc')

# Show the TF-IDF weights
for doc in tfidf[corpus]:
    print([[mydict[id], np.around(freq, decimals=2)] for id, freq in doc])
```

```
[['first', 1], ['is', 1], ['line', 1], ['the', 1], ['this', 1]]
[['is', 1], ['the', 1], ['this', 1], ['second', 1], ['sentence', 1]]
[['this', 1], ['document', 1], ['third', 1]]
[['first', 0.66], ['is', 0.24], ['line', 0.66], ['the', 0.24]]
[['is', 0.24], ['the', 0.24], ['second', 0.66], ['sentence', 0.66]]
[['document', 0.71], ['third', 0.71]]
```



```
import gensim.downloader as api
#api.info('glove-wiki-gigaword-50')
dir(api)
import gensim
dir(gensim)
```

```
['__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__path__',
 '__spec__',
 '__version__',
 'corpora',
 'downloader',
 'interfaces',
 'logger',
 'logging',
 'matutils',
 'models',
 'parsing',
 'scripts',
 'similarities',
 'summarization',
 'topic_coherence',
 'utils']
```



```
from gensim.utils import simple_preprocess
from smart_open import smart_open
import os

# Create gensim dictionary form a single text file
dictionary = corpora.Dictionary(simple_preprocess(line, deacc=True) for line in open('text8', encoding='utf-8'))
dictionary
```

```
<gensim.corpora.dictionary.Dictionary at 0x1e5d90f4c88>
```

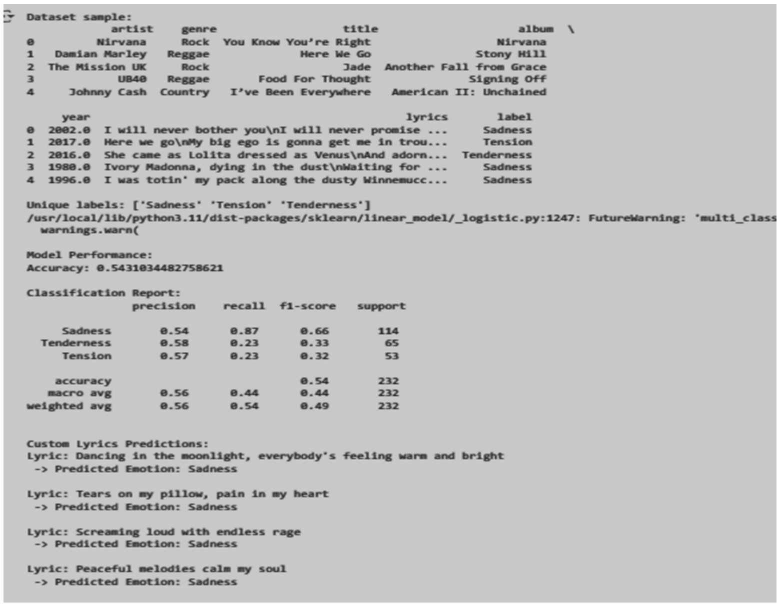
```
# doc to bag of words
```

```
# List with 2 sentences
my_docs = ["Who let the dogs out?","Who? Who? Who? Who?"]

# Tokenize the docs
tokenized_list = [simple_preprocess(doc) for doc in my_docs]

# Create the Corpus
mydict = corpora.Dictionary()
mycorpus = [mydict.doc2bow(doc, allow_update=True) for doc in tokenized_list]
pprint(mycorpus)
```

```
[[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1)], [(4, 4)]]
```

Q7. Select a application of your choice in domain like health care, banking, finance and implement

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, accuracy_score
df = pd.read_csv("SingleLabel.csv")
print("Dataset sample:\n", df.head())
print("\nUnique labels:", df['label'].unique())
X = df["lyrics"] # input text
y = df["label"]   # emotions/labels
X_train, X_test, y_train, y_test = train_test_split(
             X, y, test_size=0.2, random_state=42,stratify=y)
model = Pipeline([
    ('tfidf', TfidfVectorizer(max_features=8000, stop_words='english')),
    ('clf', LogisticRegression(max_iter=300, solver='lbfgs', multi_class='auto'))])
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("\nModel Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
sample_lyrics = [
    "Dancing in the moonlight, everybody's feeling warm and bright","Tears on my pillow, pain in my heart",
    "Screaming loud with endless rage",
    "Peaceful melodies calm my soul"]
preds = model.predict(sample_lyrics)
print("\nCustom Lyrics Predictions:")
for lyric, emotion in zip(sample_lyrics, preds):
    print(f"Lyric: {lyric}\n -> Predicted Emotion: {emotion}\n")
```



```
Dataset sample:
            artist    genre                    title                   album  \
0          Nirvana     Rock  You Know You're Right                   Nirvana
1    Damian Marley   Reggae               Here We Go              Stony Hill
2   The Mission UK     Rock                     Jade  Another Fall from Grace
3             UB40   Reggae         Food For Thought              Signing Off
4      Johnny Cash  Country    I've Been Everywhere   American II: Unchained

     year                                             lyrics       label
0  2002.0  I will never bother you\nI will never promise ...     Sadness
1  2017.0  Here we go\nMy big ego is gonna get me in trou...     Tension
2  2016.0  She came as Lolita dressed as Venus\nAnd adorn...  Tenderness
3  1980.0  Ivory Madonna, dying in the dust\nWaiting for ...     Sadness
4  1996.0  I was totin' my pack along the dusty Winnemucc...     Sadness

Unique labels: ['Sadness' 'Tension' 'Tenderness']
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class
  warnings.warn(

Model Performance:
Accuracy: 0.5431034482758621

Classification Report:
               precision    recall  f1-score   support

     Sadness       0.54      0.87      0.66       114
  Tenderness       0.58      0.23      0.33        65
     Tension       0.57      0.23      0.32        53

    accuracy                           0.54       232
   macro avg       0.56      0.44      0.44       232
weighted avg       0.56      0.54      0.49       232


Custom Lyrics Predictions:
Lyric: Dancing in the moonlight, everybody's feeling warm and bright
 -> Predicted Emotion: Sadness

Lyric: Tears on my pillow, pain in my heart
 -> Predicted Emotion: Sadness

Lyric: Screaming loud with endless rage
 -> Predicted Emotion: Sadness

Lyric: Peaceful melodies calm my soul
 -> Predicted Emotion: Sadness
```