Name : Arul kumar ARK

Roll No. : 225229103

Lab 10 :

Named Entity Recognition

Ex : 1

In [1]:
```
sentance1 = "Rajkumar said on Monday that WASHINTON -- In the wake of a string of abuses by New York poli
```

Scorce Code :

In [2]:
```
import nltk
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag
from nltk.chunk import ne_chunk
```

In [3]:
```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\arulk\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[3]: True

In [4]:
```
import nltk
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\arulk\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

Out[4]:
```
True
```

In [5]:
```
import nltk
nltk.download('maxent_ne_chunker')
```

```
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     C:\Users\arulk\AppData\Roaming\nltk_data...
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
```

Out[5]:
```
True
```

In [6]:
```
import nltk
nltk.download('words')
```

```
[nltk_data] Downloading package words to
[nltk_data]     C:\Users\arulk\AppData\Roaming\nltk_data...
[nltk_data]   Package words is already up-to-date!
```

Out[6]:
```
True
```

In [7]:
```
tokens = word_tokenize(sentance1)
tags = pos_tag(tokens)
ne_tree = ne_chunk(tags)
```

In [8]:

```
print(ne_tree)
```

```
(S
  (PERSON Rajkumar/NNP)
  said/VBD
  on/IN
  Monday/NNP
  that/IN
  (ORGANIZATION WASHINTON/NNP)
  --/:
  In/IN
  the/DT
  wake/NN
  of/IN
  a/DT
  string/NN
  of/IN
  abuses/NNS
  by/IN
  (GPE New/NNP York/NNP)
  police/NN
  officers/NNS
  in/IN
  1990s/CD
  ,/,
  (PERSON Loretta/NNP E./NNP Lynch/NNP)
  ,/,
  the/DT
  top/JJ
  frderal/JJ
  prosecutor/NN
  in/IN
  (GPE Brooklyn/NNP)
  ,/,
  spoke/VBD
  forcefully/RB
  about/IN
  thr/JJ
  pain/NN
  of/IN
  a/DT
  broken/JJ
  trust/NN
  Arican-Americans/NNPS
  felr/NN
```

```
and/CC
said/VBD
the/DT
reponibility/NN
for/IN
reparing/VBG
generations/NNS
of/IN
miscommunication/NN
and/CC
mistrust/NN
fell/VBD
to/TO
law/NN
enforcement/NN
./.)
```

Qns : 1

In [9]:
```
for chunk in ne_tree:
    if hasattr(chunk,'label'):
        print(chunk)
```

```
(PERSON Rajkumar/NNP)
(ORGANIZATION WASHINTON/NNP)
(GPE New/NNP York/NNP)
(PERSON Loretta/NNP E./NNP Lynch/NNP)
(GPE Brooklyn/NNP)
```

In [10]:
```
count_p = sum(1 for chunk in ne_tree if hasattr(chunk,'label')and chunk.label()== "PERSON")
count_l = sum(1 for chunk in ne_tree if hasattr(chunk,'label')and chunk.label()== "GPE")
count_o = sum(1 for chunk in ne_tree if hasattr(chunk,'label')and chunk.label()== "ORGANIZATION")
```

In [11]:
```
print("PERSON : ",count_p)
print("LOCATION : ",count_l)
print("ORGANIZATION :",count_o)
```

```
PERSON :  2
LOCATION :  2
ORGANIZATION : 1
```

Qns : 2

In [12]:
```
# Extract the named entities of each type
locations = []
for chunk in ne_tree:
    if hasattr(chunk, 'label'):
        if chunk.label() == 'GPE' and "police" and 'officers':
            locations.append(' '.join(c[0] for c in chunk))

# Print the named entities
print("Named entities in the input sentence:")
print("LOCATIONS:", locations)
```

```
Named entities in the input sentence:
LOCATIONS: ['New York', 'Brooklyn']
```

In [13]:
```
locations = []
for chunk in ne_tree:
    if hasattr(chunk, 'label'):
        if chunk.label() == 'GPE' and "police" and 'officers':
            locations.append(' '.join(c[0] for c in chunk))

# Check if "police officers" is recognized as a named entity
if "police officers" in locations or "police officers" in locations:
    print("Recognized")
else:
    print("Not Recognized")
```

```
Not Recognized
```

In [14]:
```
from nltk.chunk import RegexpParser
```

In [15]:
```
word = nltk.word_tokenize(sentance1)
pos_tag = nltk.pos_tag(word)
chunk = nltk.ne_chunk(pos_tag)
grammar = "NP: {<NN><NNS>}"
cp = nltk.RegexpParser(grammar)
result = cp.parse(chunk)
NE = [ " ".join(w for w, t in ele) for ele in result if isinstance(ele, nltk.Tree)]
print (NE)
```

```
['Rajkumar', 'WASHINTON', 'New York', 'police officers', 'Loretta E. Lynch', 'Brooklyn']
```

In [16]:
```
if 'police officers' in NE:
    print('The pattern "police officers" was detected in the input sentence.')
else:
    print('The pattern "police officers" was not detected in the input sentence.')
```

```
The pattern "police officers" was detected in the input sentence.
```

Qns : 3

In [17]:
```
grammar = "NP: {<DT><JJ>*<NN>}"
cp = nltk.RegexpParser(grammar)
result = cp.parse(chunk)
NE = [ " ".join(w for w, t in ele) for ele in result if isinstance(ele, nltk.Tree)]
print (NE)
```

```
['Rajkumar', 'WASHINTON', 'the wake', 'a string', 'New York', 'Loretta E. Lynch', 'the top frderal pros
ecutor', 'Brooklyn', 'a broken trust', 'the reponibility']
```

In [18]:
```
if 'the top frderal prosecutor' in NE:
    print('The pattern "the top frderal prosecutor" was detected in the input sentence.')
else:
    print('The pattern "the top frderal prosecutor" was not detected in the input sentence.')
```

```
The pattern "the top frderal prosecutor" was detected in the input sentence.
```

Ex : 2

In [19]:
```
sentance2 = "European authorities fines Google a record $5.1 billion on Wednesday for abusing its power i
```

Qns : 1

In [20]:
```
word = nltk.word_tokenize(sentance2)
pos_tag = nltk.pos_tag(word)
chunk = nltk.ne_chunk(pos_tag)
grammar = "NP: {<CD>}"
cp = nltk.RegexpParser(grammar)
result = cp.parse(chunk)
NE = [ " ".join(w for w, t in ele) for ele in result if isinstance(ele, nltk.Tree)]
print (NE)
```

```
['European', '5.1', 'billion']
```

In [21]:
```
if 'European'and '5.10'and 'billion' in NE:
    print('The BLOD NE sentence was detected.')
else:
    print('The BLOD NE sentence was not detected..')
```

```
The BLOD NE sentence was detected.
```

In [22]:
```
grammar = "CD:{<\$><CD>}"
cp = nltk.RegexpParser(grammar)
result = cp.parse(chunk)
NE = [ " ".join(w for w, t in ele) for ele in result if isinstance(ele, nltk.Tree)]
print (NE)
```

```
['European', '$ 5.1']
```

In [23]:
```
if  '$ 5.1' in NE:
    print('The $ 5.1 sentence was detected.')
else:
    print('The $ 5.1 sentence was not detected.')
```

```
The $ 5.1 sentence was detected.
```

In [24]:
```
grammar = "NP: {<DT><JJ>*<NN>}"
cp = nltk.RegexpParser(grammar)
result = cp.parse(chunk)
NE = [ " ".join(w for w, t in ele) for ele in result if isinstance(ele, nltk.Tree)]
print (NE)
```

```
['European', 'a record', 'the moblie', 'the company']
```

In [25]:
```
if  'the moblie' and 'the company' in NE:
    print("The 'the moblie' and 'the company' sentence was detected.")
else:
    print("The 'the moblie' and 'the company' sentence was NOT detected.")
```

```
The 'the moblie' and 'the company' sentence was detected.
```

Ex : 3

In [27]:

```
import nltk
from collections import Counter

with open('foodrec.txt', 'r') as file:
    text = file.read()
    tags = nltk.ne_chunk(nltk.pos_tag(nltk.word_tokenize(text)))
    print('Named Entity Frequencies:')
    print('------------------------')
    for chunk in tags:
        if hasattr(chunk, 'label') and chunk.label() != 'S':  # Check if chunk is a named entity chunk
            label, entity = zip(*chunk)
            print(label[0] + ':', Counter(entity))
    file.close()
```

```
Named Entity Frequencies:
------------------------
BEEF: Counter({'NNP': 1})
TENDERLOIN: Counter({'NNP': 1})
Sebeef: Counter({'NNP': 1})
```

In [ ]: