NATURAL LANGUAGE PROCESSING

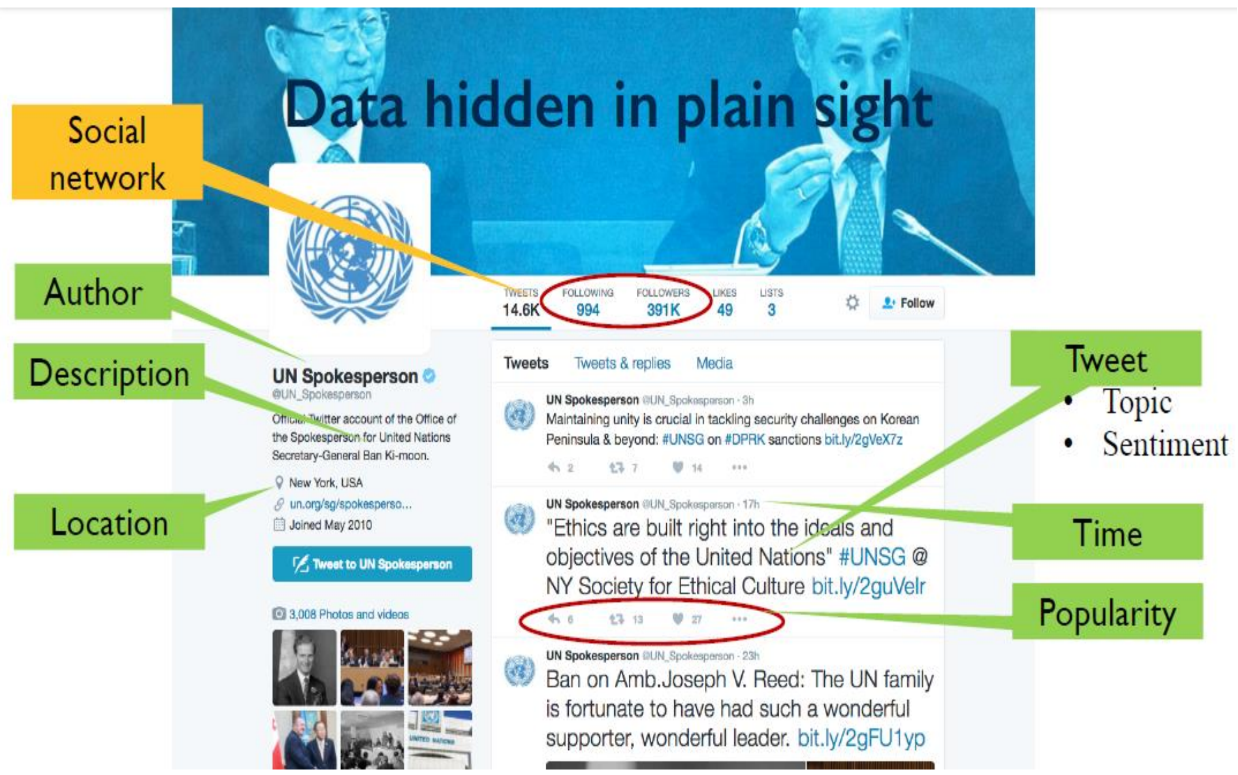
Arpendu Ganguly

NATURAL LANGUAGE PROCESSING

- **Natural language processing (NLP) is any computation, manipulation of natural language**
- Natural Language
    - Language used for everyday communication by humans
    - Evolves with time
- **Text data is growing fast!**
    - Data continues to grow exponentially
    - Approximately 80% of all data is estimated to be unstructured, text-rich data

NATURAL LANGUAGE PROCESSING

- Parse text
- Find / Identify / Extract relevant information from text
- Classify text documents
- Search for relevant text documents
- Sentiment analysis
- Topic modeling



UN Spokesperson
@UN_Spokesperson
Official Twitter account of the Office of the Spokesperson for United Nations Secretary-General Ban Ki-moon.
New York, USA
un.org/sg/spokesperso...
Joined May 2010

UN Spokesperson @UN_Spokesperson · 3h
Maintaining unity is crucial in tackling security challenges on Korean Peninsula & beyond: #UNSG on #DPRK sanctions bit.ly/2gVeX7z
2 7 14

UN Spokesperson @UN_Spokesperson · 17h
"Ethics are built right into the ideals and objectives of the United Nations" #UNSG @ NY Society for Ethical Culture bit.ly/2guVelr
6 13 27

UN Spokesperson @UN_Spokesperson · 23h
Ban on Amb.Joseph V. Reed: The UN family is fortunate to have had such a wonderful supporter, wonderful leader. bit.ly/2gFU1yp

NPL TASKS

NLP Tasks: A Broad Spectrum

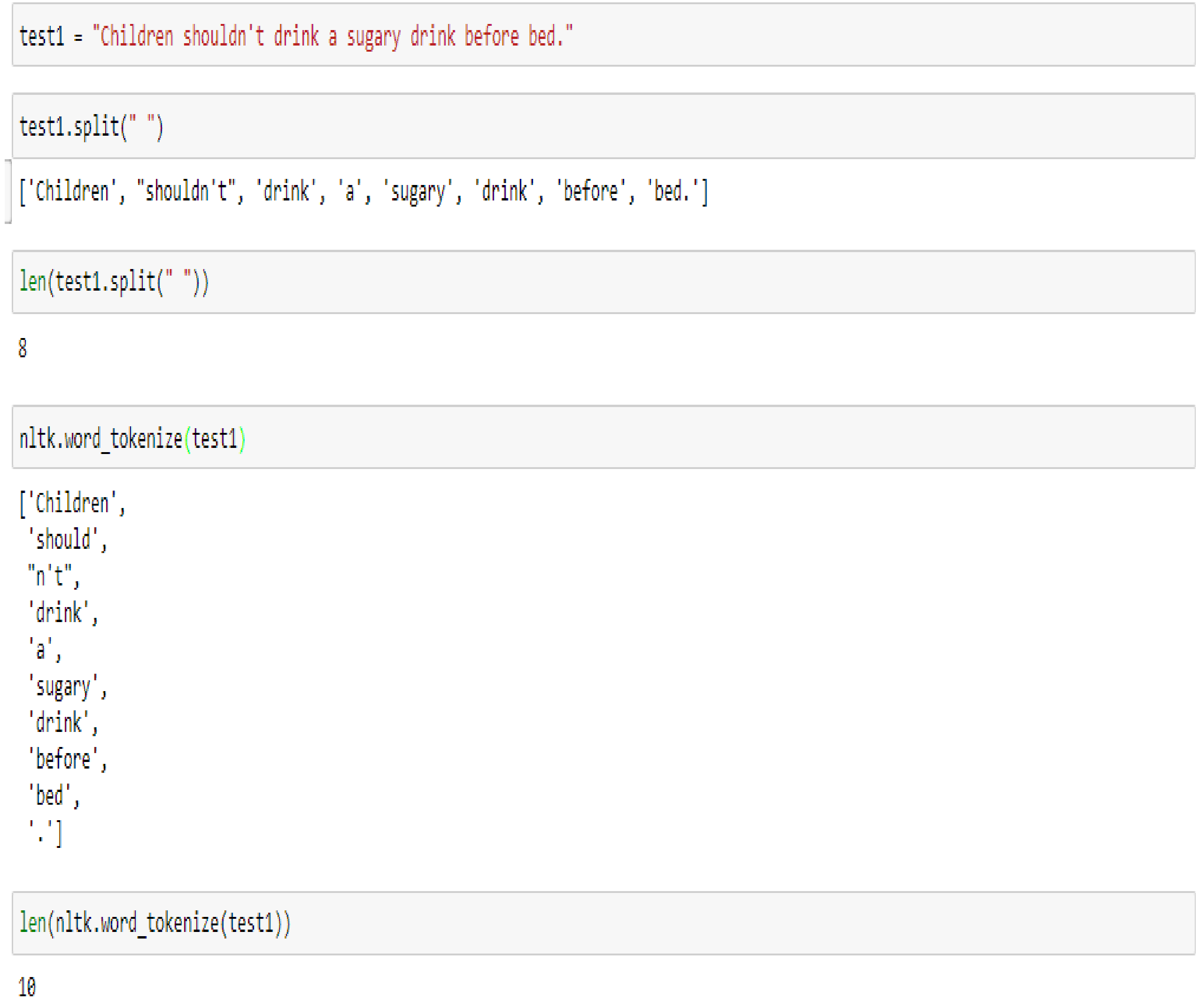
- Counting words, counting frequency of words
- Finding sentence boundaries
- Part of speech tagging
- Parsing the sentence structure
- Text Classification, Identifying semantic roles
- Identifying entities in a sentence / Named Entity Recognition
- Finding which pronoun refers to which entity / Co-reference and pronoun resolution
- much more …

NLTK: Natural Language Toolkit

- Open source library in Python
- Has support for most NLP tasks
- Also provides access to numerous text corpora

TOKENIZATION

Splitting a sentence into words / tokens



```
test1 = "Children shouldn't drink a sugary drink before bed."
```

```
test1.split(" ")
```

```
['Children', "shouldn't", 'drink', 'a', 'sugary', 'drink', 'before', 'bed.']
```

```
len(test1.split(" "))
```

```
8
```

```
nltk.word_tokenize(test1)
```

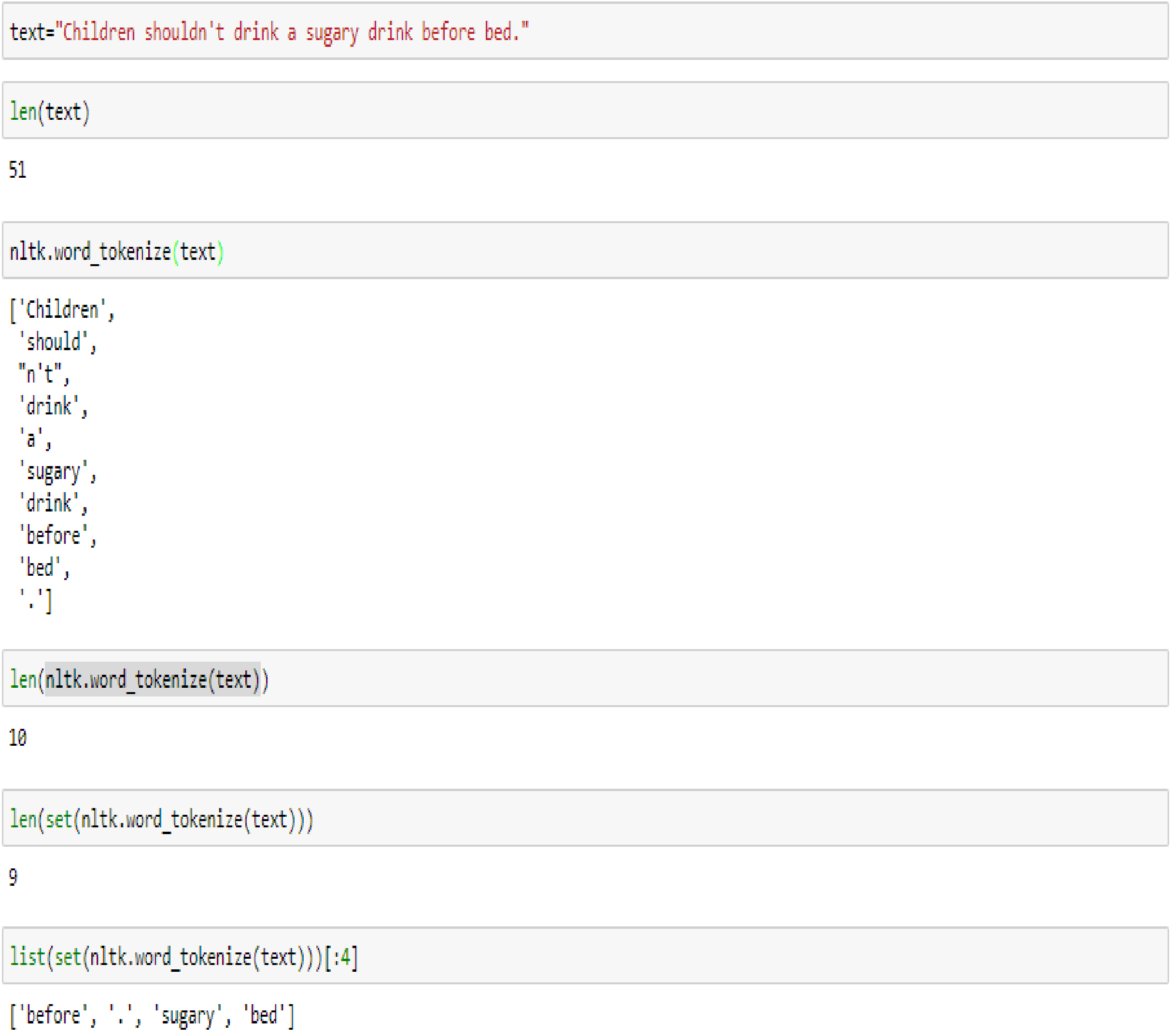
```
['Children',
 'should',
 "n't",
 'drink',
 'a',
 'sugary',
 'drink',
 'before',
 'bed',
 '.']
```

```
len(nltk.word_tokenize(test1))
```

```
10
```

COUNTING VOCABULARY OF WORDS

Count unique words



```
text="Children shouldn't drink a sugary drink before bed."
```

```
len(text)
```

```
51
```

```
nltk.word_tokenize(text)
```

```
['Children',
 'should',
 "n't",
 'drink',
 'a',
 'sugary',
 'drink',
 'before',
 'bed',
 '.']
```

```
len(nltk.word_tokenize(text))
```

```
10
```

```
len(set(nltk.word_tokenize(text)))
```

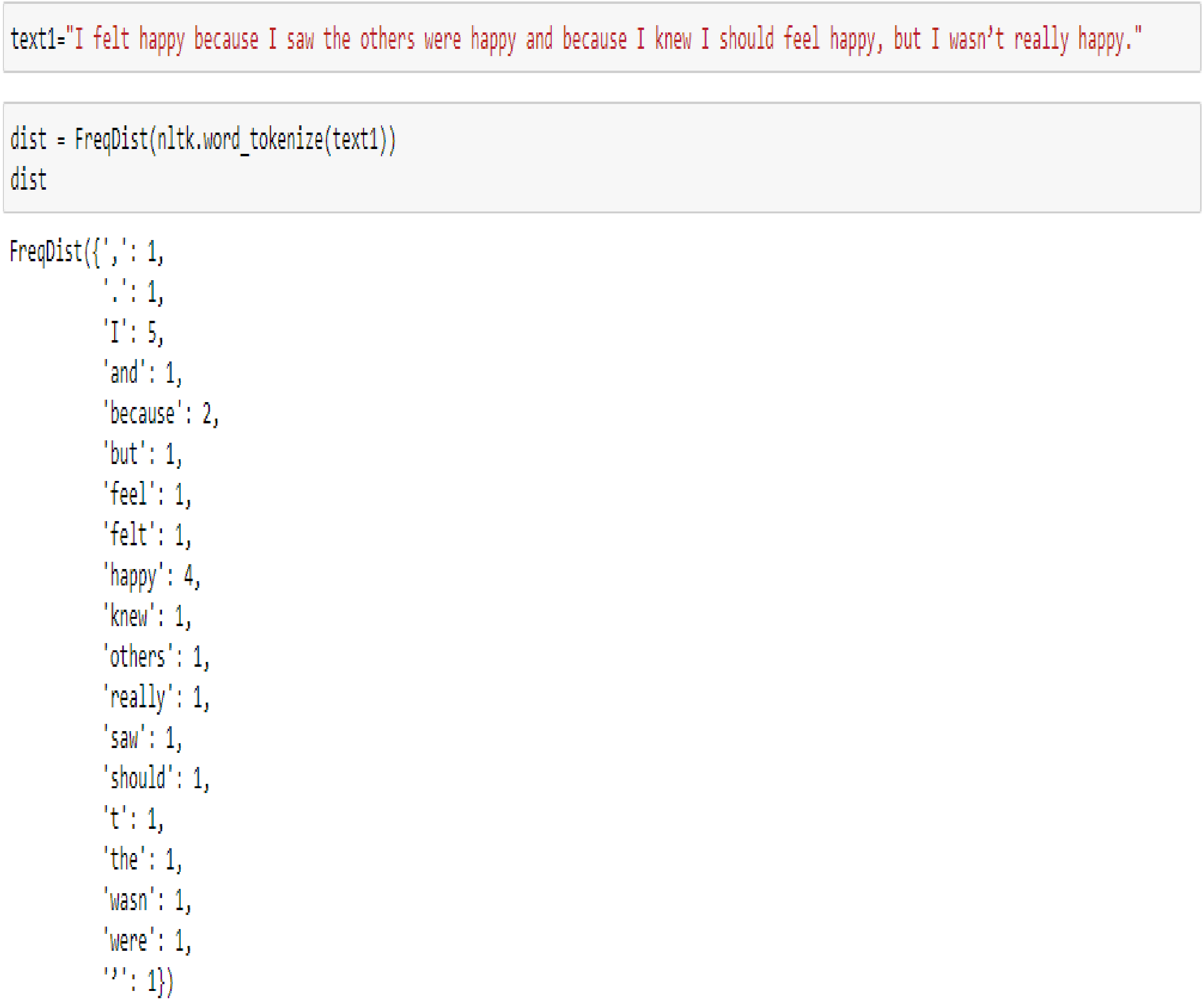
```
9
```

```
list(set(nltk.word_tokenize(text)))[:4]
```

```
['before', '.', 'sugary', 'bed']
```

FREQUENCY OF WORDS

Mapping of unique word to count



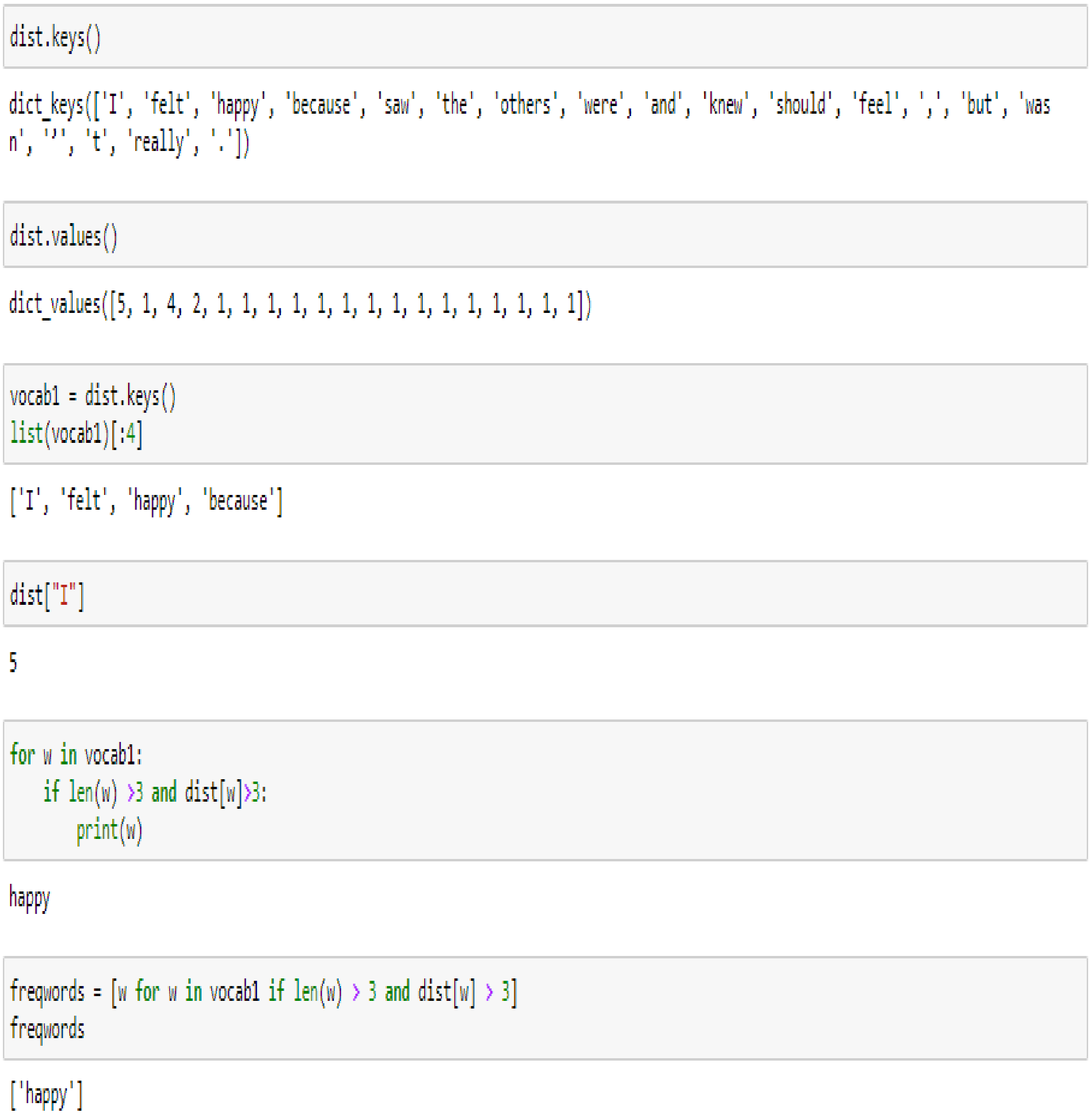
```
text1="I felt happy because I saw the others were happy and because I knew I should feel happy, but I wasn’t really happy."
```

```
dist = FreqDist(nltk.word_tokenize(text1))
dist
```

```
FreqDist({',': 1,
          '.': 1,
          'I': 5,
          'and': 1,
          'because': 2,
          'but': 1,
          'feel': 1,
          'felt': 1,
          'happy': 4,
          'knew': 1,
          'others': 1,
          'really': 1,
          'saw': 1,
          'should': 1,
          't': 1,
          'the': 1,
          'wasn': 1,
          'were': 1,
          '’': 1})
```

FREQUENCY OF WORDS

Mapping of unique word to count



```
dist.keys()
```

```
dict_keys(['I', 'felt', 'happy', 'because', 'saw', 'the', 'others', 'were', 'and', 'knew', 'should', 'feel', ',', 'but', 'was
n', '’', 't', 'really', '.'])
```

```
dist.values()
```

```
dict_values([5, 1, 4, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

```
vocab1 = dist.keys()
list(vocab1)[:4]
```

```
['I', 'felt', 'happy', 'because']
```

```
dist["I"]
```

```
5
```

```
for w in vocab1:
    if len(w) >3 and dist[w]>3:
        print(w)
```

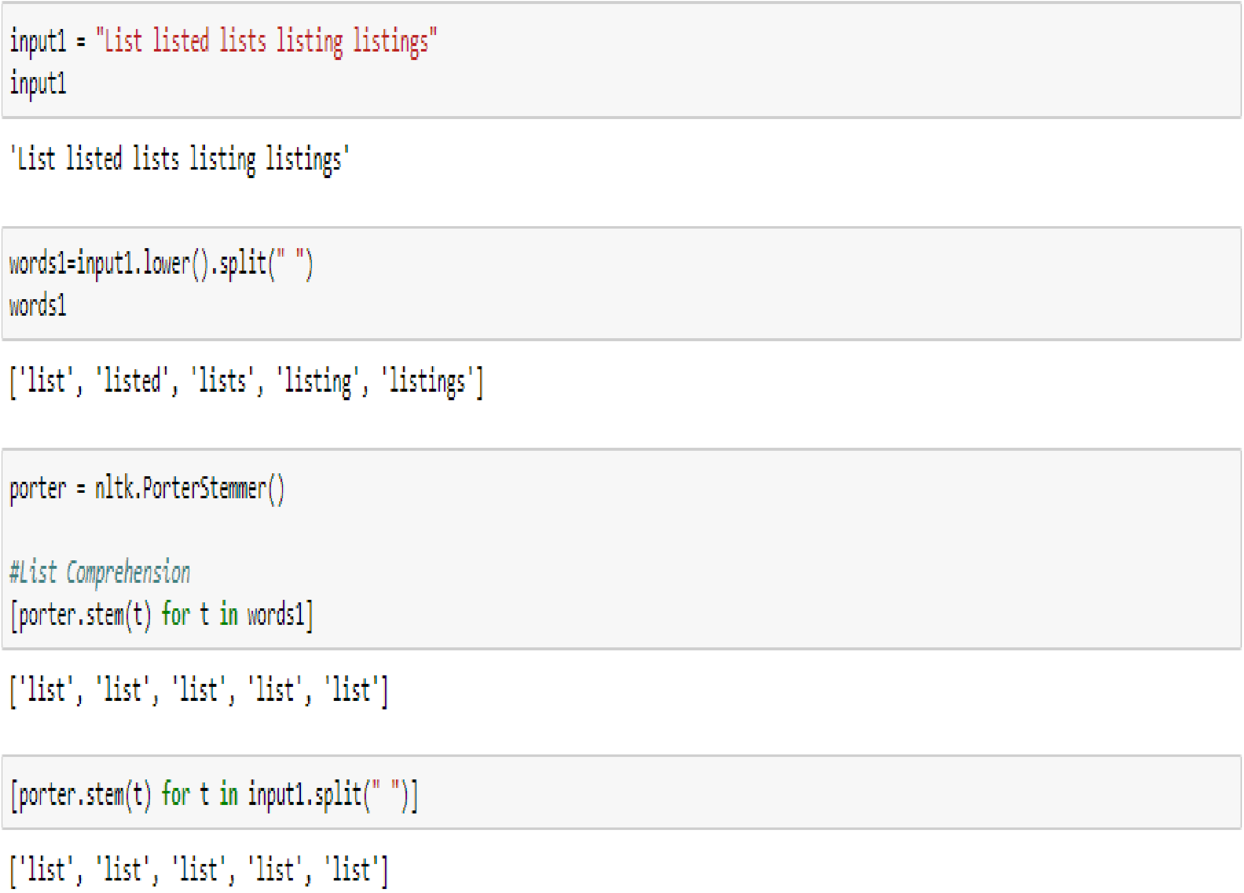
```
happy
```

```
freqwords = [w for w in vocab1 if len(w) > 3 and dist[w] > 3]
freqwords
```

```
['happy']
```

NORMALIZATION AND STEMMING

Normalization involves eliminating punctuation, converting the entire text into lowercase or uppercase and so on.



```
input1 = "List listed lists listing listings"
input1
```

```
'List listed lists listing listings'
```

```
words1=input1.lower().split(" ")
words1
```

```
['list', 'listed', 'lists', 'listing', 'listings']
```

```
porter = nltk.PorterStemmer()

#List Comprehension
[porter.stem(t) for t in words1]
```

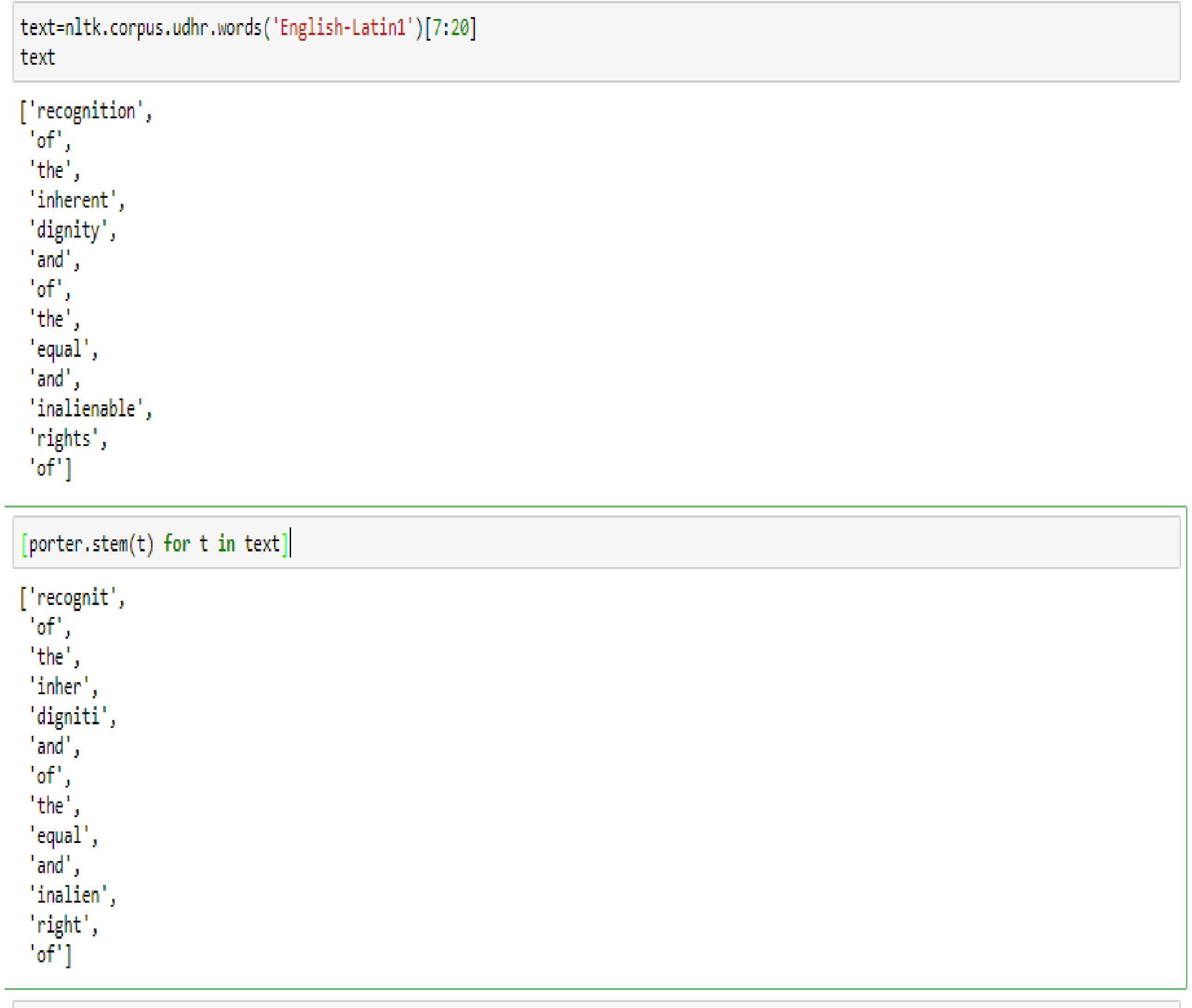
```
['list', 'list', 'list', 'list', 'list']
```

```
[porter.stem(t) for t in input1.split(" ")]
```

```
['list', 'list', 'list', 'list', 'list']
```

STEMMING

Reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.



```
text=nltk.corpus.udhr.words('English-Latin1')[7:20]
text
```

```
['recognition',
 'of',
 'the',
 'inherent',
 'dignity',
 'and',
 'of',
 'the',
 'equal',
 'and',
 'inalienable',
 'rights',
 'of']
```

```
[porter.stem(t) for t in text]
```

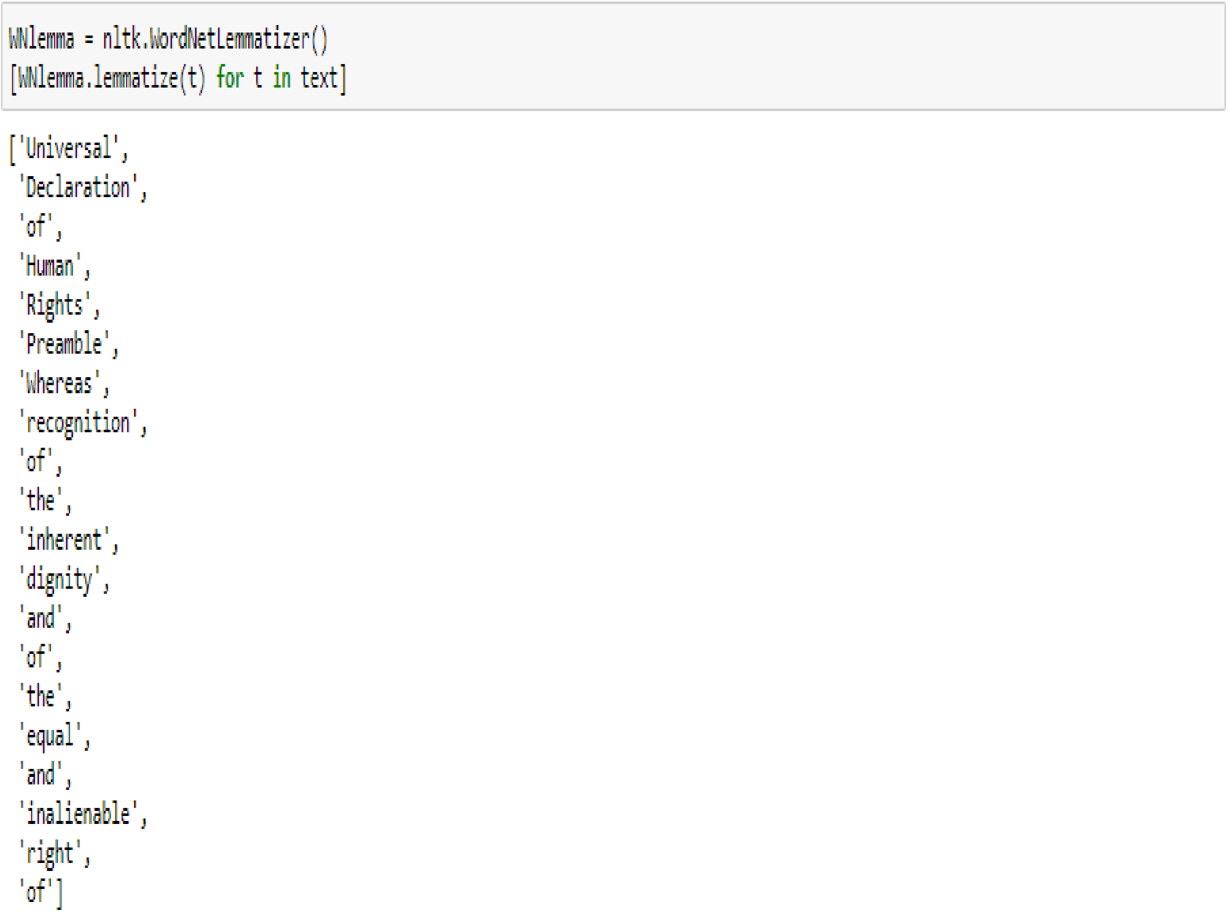
```
['recognit',
 'of',
 'the',
 'inher',
 'digniti',
 'and',
 'of',
 'the',
 'equal',
 'and',
 'inalien',
 'right',
 'of']
```

STEMMING AND LEMMATIZATION

Stemming and lemmatization

reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.

Lemmatization: Stemming, but resulting stems are all valid words.



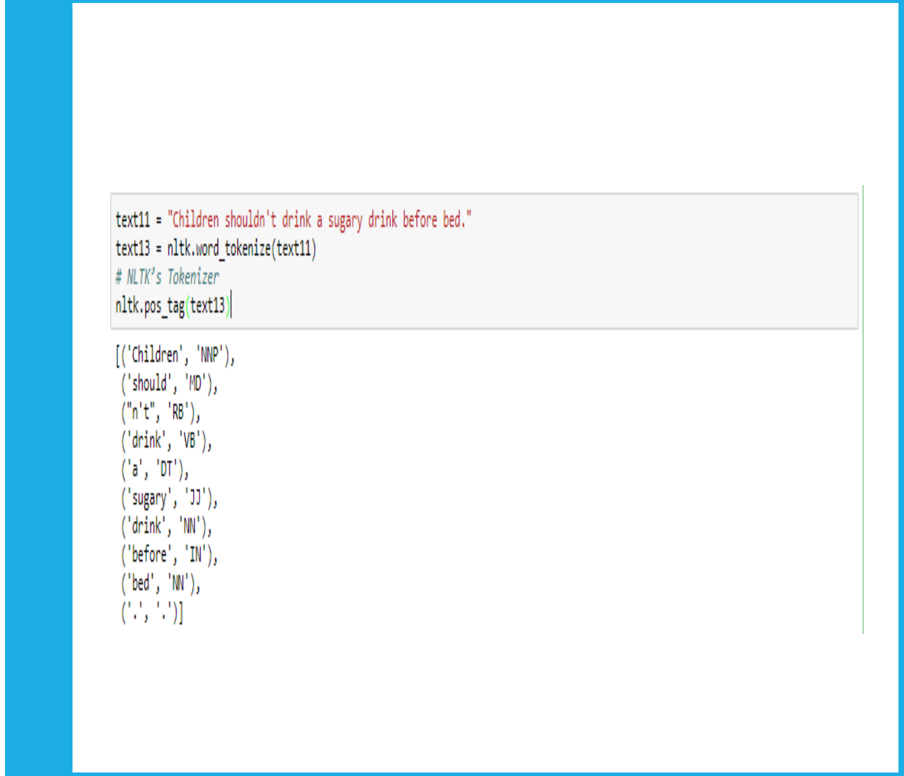
```
WNlemma = nltk.WordNetLemmatizer()
[WNlemma.lemmatize(t) for t in text]
```

```
['Universal',
 'Declaration',
 'of',
 'Human',
 'Rights',
 'Preamble',
 'Whereas',
 'recognition',
 'of',
 'the',
 'inherent',
 'dignity',
 'and',
 'of',
 'the',
 'equal',
 'and',
 'inalienable',
 'right',
 'of']
```

PART-OF-SPEECH (POS) TAGGING

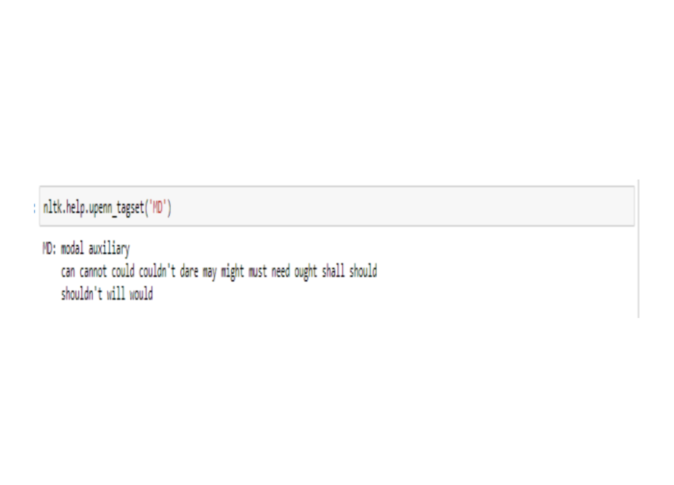
Part of speech tagging

- identification of words as nouns, verbs, adjectives, adverbs, etc
- Many more tags or word classes than just these



```
text11 = "Children shouldn't drink a sugary drink before bed."
text13 = nltk.word_tokenize(text11)
# NLTK's Tokenizer
nltk.pos_tag(text13)
```

```
[('Children', 'NNP'),
 ('should', 'MD'),
 ("n't", 'RB'),
 ('drink', 'VB'),
 ('a', 'DT'),
 ('sugary', 'JJ'),
 ('drink', 'NN'),
 ('before', 'IN'),
 ('bed', 'NN'),
 ('.', '.')]
```



```
nltk.help.upenn_tagset('MD')
```

```
MD: modal auxiliary
    can cannot could couldn't dare may might must need ought shall should
    shouldn't will would
```

TEXT FEATURE EXTRACTION

Tf–idf transfom/term weighting

- Often words occurring frequently (e.g. “the”, “a”, “is” in English) carry very little meaningful information about the actual contents of the document.
- Weights high to terms which are rarer yet more interesting
- **tf–idf** means **term-frequency times inverse document-frequency**
- **Tf** means **term-frequency,** the number of times a term occurs in a given document
- **Inverse document-frequency**

- where is the total number of documents, and is the number of documents that contain term t.

TEXT FEATURE EXTRACTION

Bag of Words representation

- ***tokenizing*** *strings* and giving an integer id for each possible token, for instance by using white-spaces and punctuation as token separators
- ***counting*** the occurrences of tokens in each document
- ***normalizing*** *and weighting* with diminishing importance tokens that occur in the majority of samples / documents

Sparse matrix

- sparse matrix or sparse array is a matrix in which most of the elements are zero

***n*-gram**

- A contiguous sequence of *n* items from a given sample of text or speech
- Example
  - "I am working in Accenture."
  - Unigram (1 gram): "I", "am", "working", "in", "Accenture"
  - Bigram (2 gram) : "I am", "am working", "working in", "in Accenture"