NLP

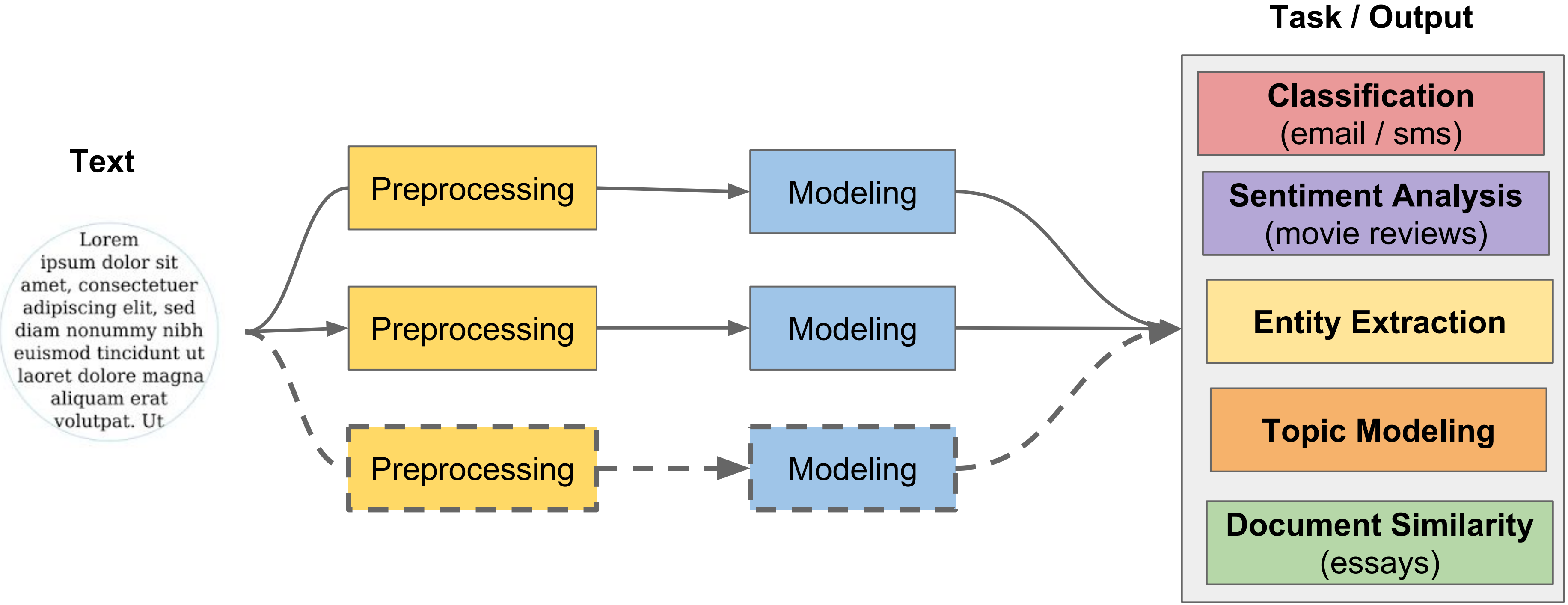
By: Esraa Madi

What is NLP:

Allows computers to **understand**, **analyze** and **extract information** from human language.

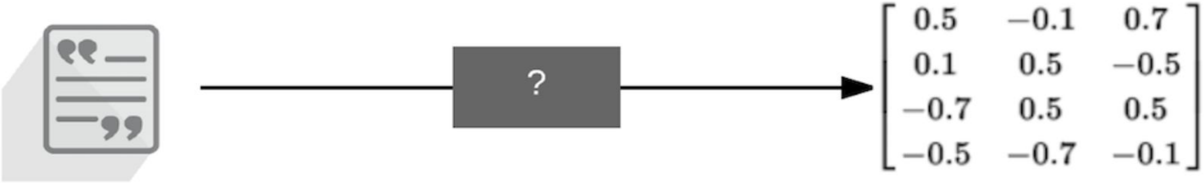
How does it work : NLP Pipeline

It's a technique to streamline the nlp process into stages


Text
Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoret dolore magna aliquam erat volutpat. Ut
Preprocessing
Modeling
Preprocessing
Modeling
Preprocessing
Modeling
Task / Output
Classification (email / sms)
Sentiment Analysis (movie reviews)
Entity Extraction
Topic Modeling
Document Similarity (essays)

Stage 1 : Preprocessing

Putting our data **(text)** in the proper format **(numerical vectors)** to perform machine learning



$$\begin{bmatrix} 0.5 & -0.1 & 0.7 \\ 0.1 & 0.5 & -0.5 \\ -0.7 & 0.5 & 0.5 \\ -0.5 & -0.7 & -0.1 \end{bmatrix}$$

Preprocessing can involve as little or as many steps as **needed** / **required** / **wanted**

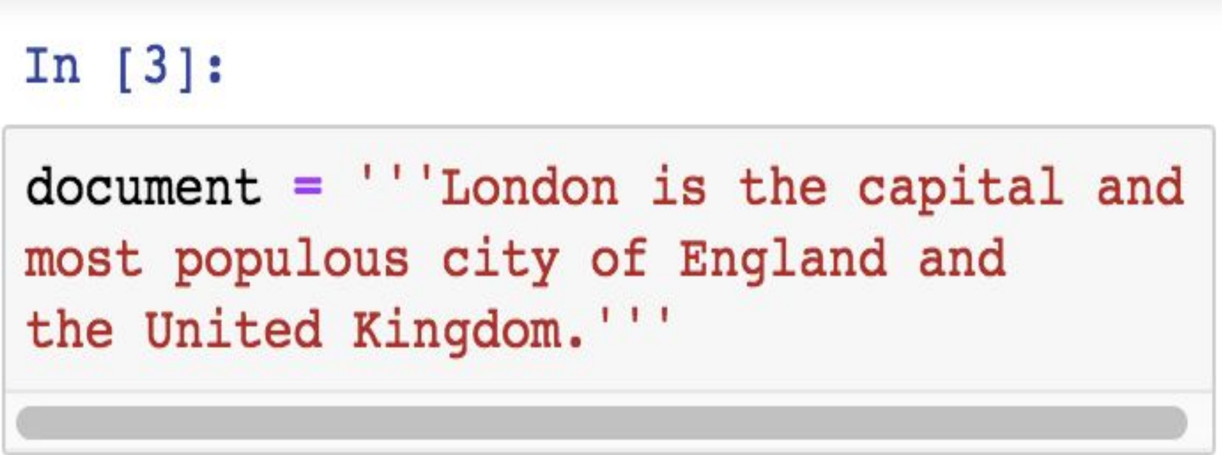
Stage 1 : Preprocessing

NLP libraries used in preprocessing text :

1- Scikit learn

2- Natural language toolkit (NLTK)

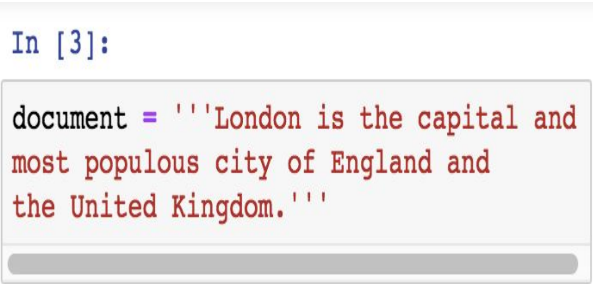
Stage 1 : Preprocessing



```
document = '''London is the capital and
most populous city of England and
the United Kingdom.'''
```

Preprocessing step 1 : tokenization

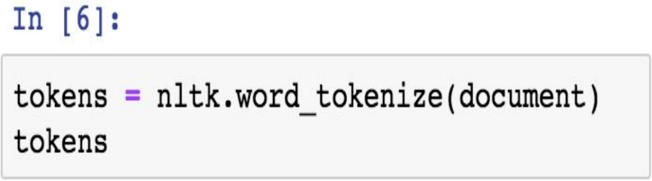
Splitting the **document** to **words (Tokens)** can be accessed.



```
In [3]:
document = '''London is the capital and
most populous city of England and
the United Kingdom.'''
```

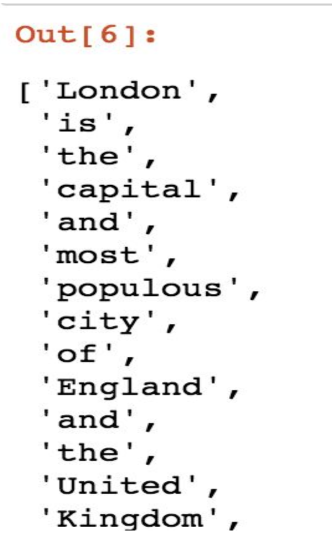
Document

Tokenization



```
In [6]:
tokens = nltk.word_tokenize(document)
tokens
```

Tokens

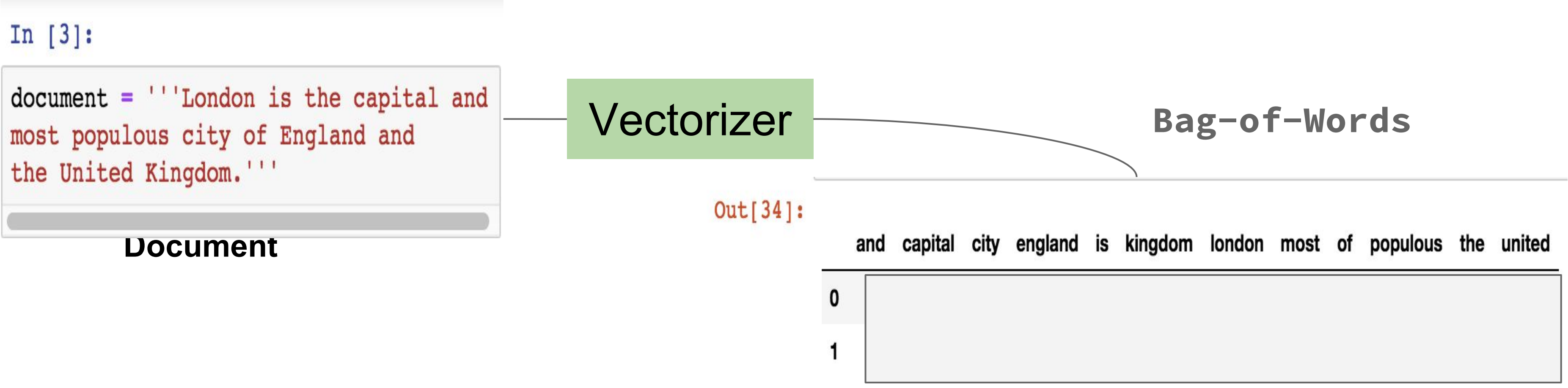


```
Out[6]:
['London',
 'is',
 'the',
 'capital',
 'and',
 'most',
 'populous',
 'city',
 'of',
 'England',
 'and',
 'the',
 'United',
 'Kingdom',
```

Preprocessing step 2 : Vectorization

Transform **documents** (text) to the **Bag-of-Words** model.

1. Splitting the documents into tokens
2. Assigning a weight to each token proportional to the frequency with which it shows up in the document.



In [3]:

```
document = '''London is the capital and
most populous city of England and
the United Kingdom.'''
```

Document

Vectorizer

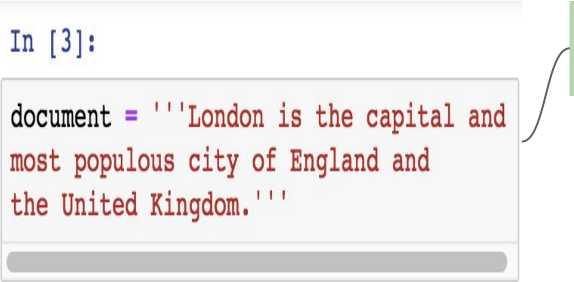
Bag-of-Words

Out[34]:

	and	capital	city	england	is	kingdom	london	most	of	populous	the	united
0												
1												

Preprocessing step 2: Vectorization - Countvectorizer

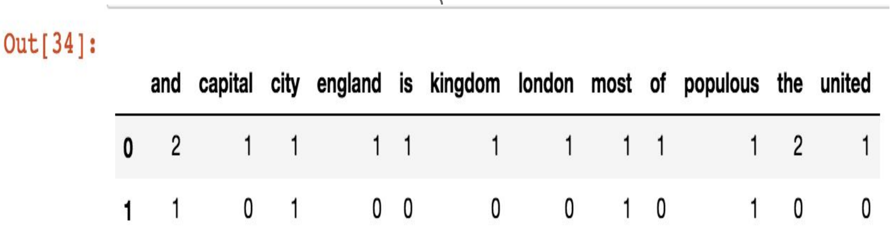
Token weight = Counts the number of times a token shows up in the document



In [3]:

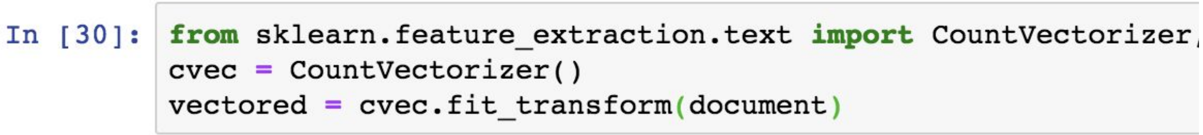
```
document = '''London is the capital and
most populous city of England and
the United Kingdom.'''
```

Countvectorizer



Out[34]:

	and	capital	city	england	is	kingdom	london	most	of	populous	the	united
0	2	1	1	1	1	1	1	1	1	1	2	1
1	1	0	1	0	0	0	0	1	0	1	0	0

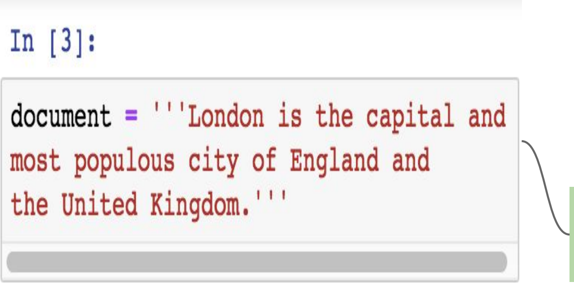


In [30]:

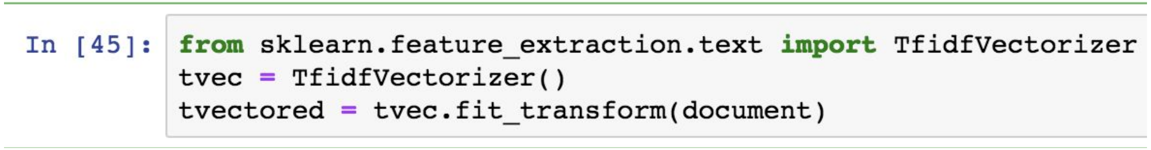
```
from sklearn.feature_extraction.text import CountVectorizer,
cvec = CountVectorizer()
vectored = cvec.fit_transform(document)
```

Preprocessing step 2: Vectorization- TF_IDF Vectorizer

- Term frequency-inverse document frequency
- Token weight depends on its frequency in a document and how common that token is across all documents.
- it basically reduces values of common word that are used in different document.

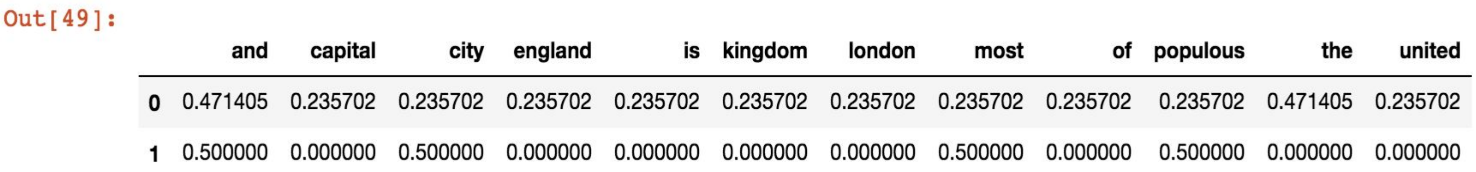


```
In [3]:

document = '''London is the capital and
most populous city of England and
the United Kingdom.'''
```



```
In [45]: from sklearn.feature_extraction.text import TfidfVectorizer
         tvec = TfidfVectorizer()
         tvectored = tvec.fit_transform(document)
```

TF-IDF Vectorizer

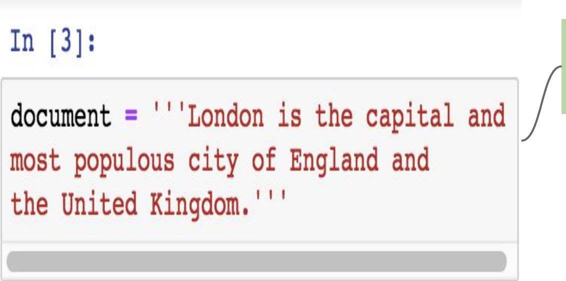


Out[49]:

	and	capital	city	england	is	kingdom	london	most	of	populous	the	united
0	0.471405	0.235702	0.235702	0.235702	0.235702	0.235702	0.235702	0.235702	0.235702	0.235702	0.471405	0.235702
1	0.500000	0.000000	0.500000	0.000000	0.000000	0.000000	0.000000	0.500000	0.000000	0.500000	0.000000	0.000000

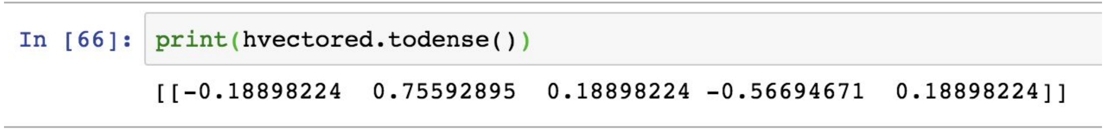
Preprocessing step 2: Vectorization - Hash Vectorizer

- Use a one way hash of words to convert them to integers.
- No vocabulary is required.
- The Hash is a one-way function so there is no way to convert the encoding back to a word

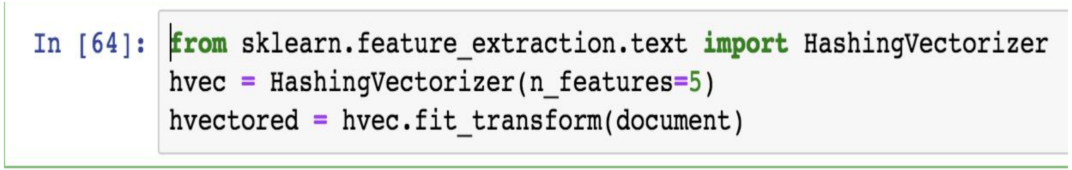


```
In [3]:
document = '''London is the capital and
most populous city of England and
the United Kingdom.'''
```

Hash Vectorizer



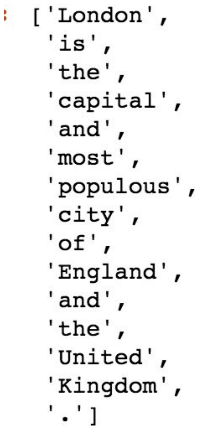
```
In [66]: print(hvectored.todense())

         [[-0.18898224  0.75592895  0.18898224 -0.56694671  0.18898224]]
```



```
In [64]: from sklearn.feature_extraction.text import HashingVectorizer
         hvec = HashingVectorizer(n_features=5)
         hvectored = hvec.fit_transform(document)
```

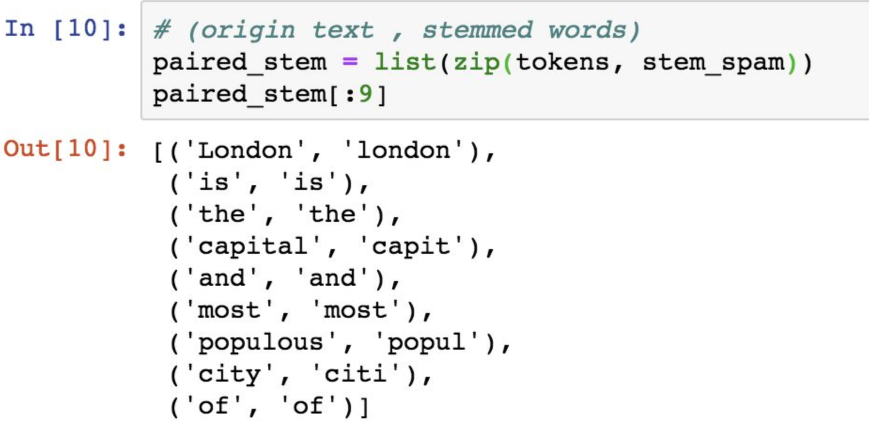
Preprocessing step 3 : Stemming

- Reduce words to a stem (root) form.
- It bundles together words of same root.
- Ex: It bundles "response" and "respond" into a common "respon"

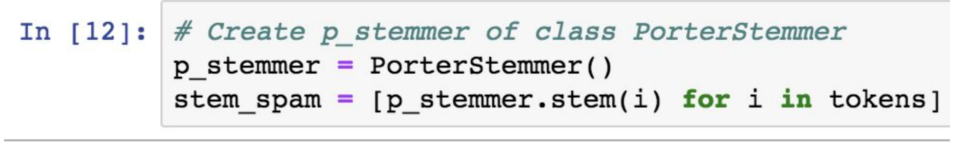


```
['London',
 'is',
 'the',
 'capital',
 'and',
 'most',
 'populous',
 'city',
 'of',
 'England',
 'and',
 'the',
 'United',
 'Kingdom',
 '.']
```

Tokens



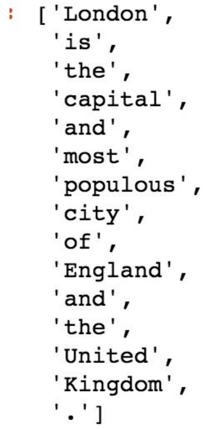
```
In [10]: # (origin text , stemmed words)
         paired_stem = list(zip(tokens, stem_spam))
         paired_stem[:9]

Out[10]: [('London', 'london'),
          ('is', 'is'),
          ('the', 'the'),
          ('capital', 'capit'),
          ('and', 'and'),
          ('most', 'most'),
          ('populous', 'popul'),
          ('city', 'citi'),
          ('of', 'of')]
```



```
In [12]: # Create p_stemmer of class PorterStemmer
         p_stemmer = PorterStemmer()
         stem_spam = [p_stemmer.stem(i) for i in tokens]
```

Preprocessing step 4 : lemmatizing

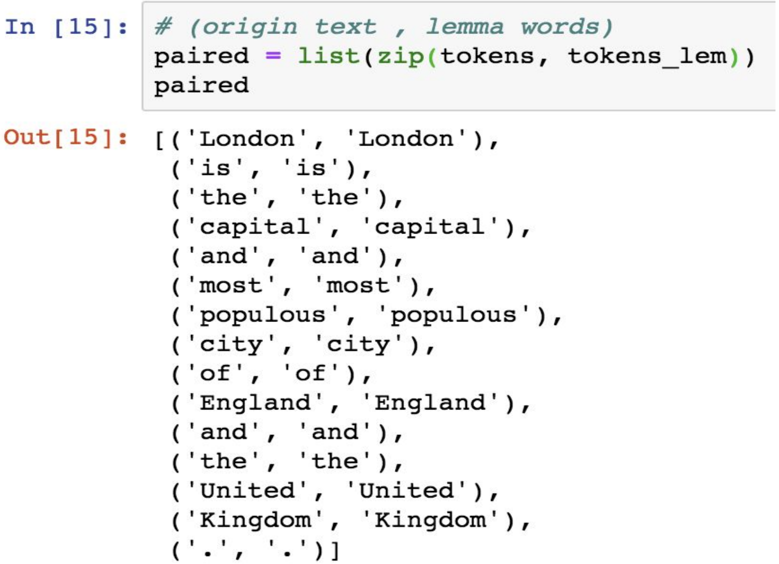
Word lemmatizing is similar to stemming, but the difference is the result of lemmatizing is a real word.



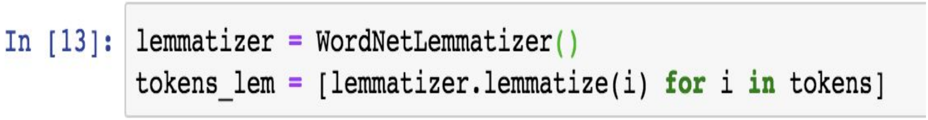
```
: ['London',
   'is',
   'the',
   'capital',
   'and',
   'most',
   'populous',
   'city',
   'of',
   'England',
   'and',
   'the',
   'United',
   'Kingdom',
   '.']
```

Tokens

Lemmatizing



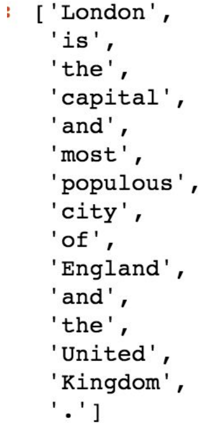
```
In [15]: # (origin text , lemma words)
         paired = list(zip(tokens, tokens_lem))
         paired

Out[15]: [('London', 'London'),
          ('is', 'is'),
          ('the', 'the'),
          ('capital', 'capital'),
          ('and', 'and'),
          ('most', 'most'),
          ('populous', 'populous'),
          ('city', 'city'),
          ('of', 'of'),
          ('England', 'England'),
          ('and', 'and'),
          ('the', 'the'),
          ('United', 'United'),
          ('Kingdom', 'Kingdom'),
          ('.', '.')]
```



```
In [13]: lemmatizer = WordNetLemmatizer()
         tokens_lem = [lemmatizer.lemmatize(i) for i in tokens]
```

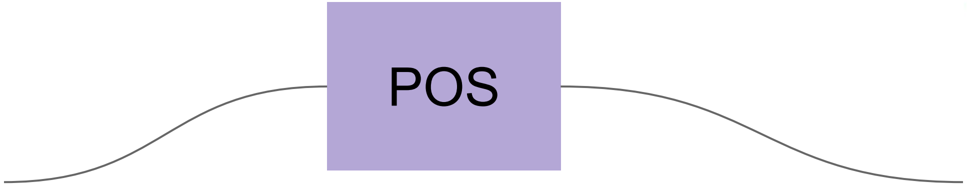
Preprocessing step 5 : part of speech (POS) tagging

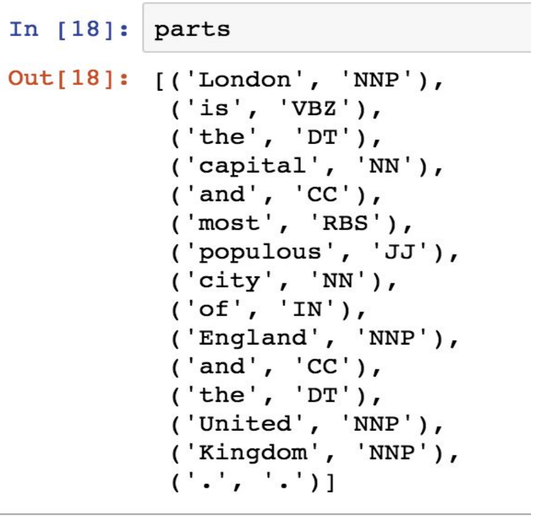
For each token try to guess its part of speech – whether it is a noun, a verb, an adjective and so on.



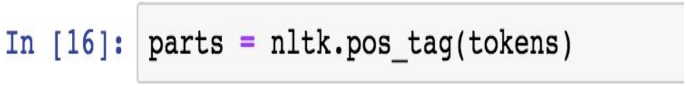
```
: ['London',
   'is',
   'the',
   'capital',
   'and',
   'most',
   'populous',
   'city',
   'of',
   'England',
   'and',
   'the',
   'United',
   'Kingdom',
   '.']
```

Tokens





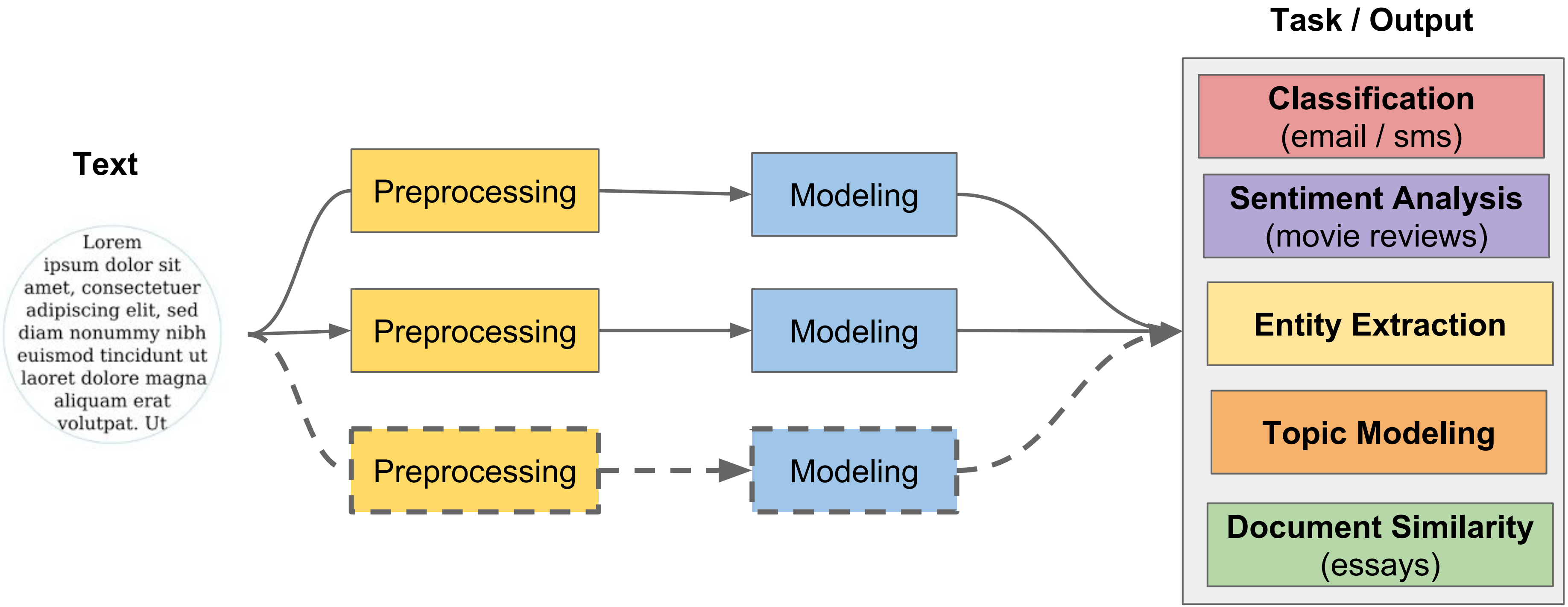
```
In [18]: parts

Out[18]: [('London', 'NNP'),
          ('is', 'VBZ'),
          ('the', 'DT'),
          ('capital', 'NN'),
          ('and', 'CC'),
          ('most', 'RBS'),
          ('populous', 'JJ'),
          ('city', 'NN'),
          ('of', 'IN'),
          ('England', 'NNP'),
          ('and', 'CC'),
          ('the', 'DT'),
          ('United', 'NNP'),
          ('Kingdom', 'NNP'),
          ('.', '.')]
```



```
In [16]: parts = nltk.pos_tag(tokens)
```

How is it work : NLP Pipeline

It's a technique to streamline the nlp process into stages


Task / Output
Text
Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoret dolore magna aliquam erat volutpat. Ut
Preprocessing
Modeling
Preprocessing
Modeling
Preprocessing
Modeling
Classification (email / sms)
Sentiment Analysis (movie reviews)
Entity Extraction
Topic Modeling
Document Similarity (essays)

Stage 2 : modeling

Supervised

Naive Bayes, SVM, Linear regression, K-NN neighnors

Unsupervised

K-means