THINKFUL

Text Data Cleaning and Preprocessing

DATA SCIENCE

Warm Up

- What do you anticipate is involved with preprocessing and cleaning text data?

Agenda

- Text Data Preprocessing
- Deconstructing Documents
- Sentence Tokenization
- Word Tokenization
- Part of Speech Tagging
- Cleaning Text Data
- Making Characters Lowercase
- Stop Word Removal
- Normalization

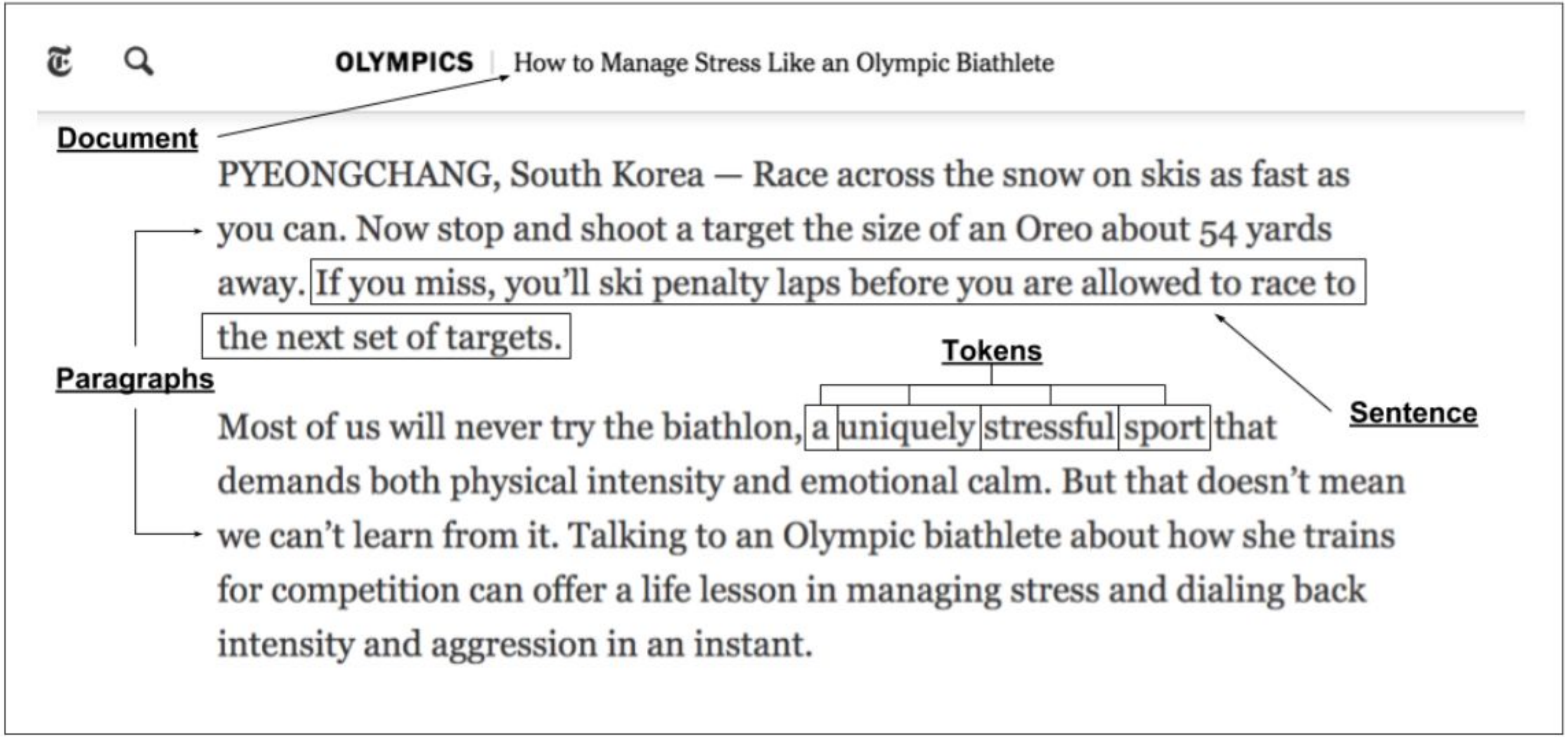
Text Data Preprocessing

- Before we can extract any information or perform any analytics on our text data, we need to preprocess it.
- Preprocessing will help us deconstruct our documents into their components so that we can better identify the information we are looking for and convert text into numbers that can be computed upon.
- The major tasks involved in preprocessing include tokenization, part of speech tagging, and cleaning text.

Deconstructing Documents

- The first part of preprocessing involves tokenizing our documents or deconstructing them into the components they are comprised of.
- Paragraphs - units of document structure.
- Sentences - units of discourse.
- Tokens - syntactic units of language.

Deconstructing Documents



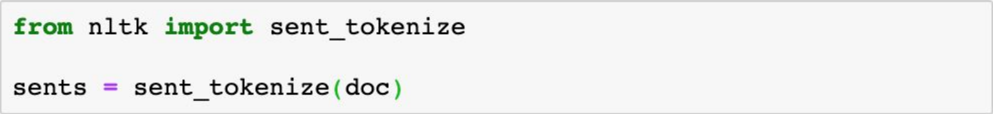
OLYMPICS | How to Manage Stress Like an Olympic Biathlete

PYEONGCHANG, South Korea — Race across the snow on skis as fast as you can. Now stop and shoot a target the size of an Oreo about 54 yards away. If you miss, you'll ski penalty laps before you are allowed to race to the next set of targets.

Most of us will never try the biathlon, a uniquely stressful sport that demands both physical intensity and emotional calm. But that doesn't mean we can't learn from it. Talking to an Olympic biathlete about how she trains for competition can offer a life lesson in managing stress and dialing back intensity and aggression in an instant.

Sentence Tokenization

- Identifying and extracting the individual sentences in a document, typically done with the use of a trained tokenizer.
- NLTK has a `sent_tokenize` function that can help us do this.
- Once we have imported it, we can just call the `sent_tokenize` function and pass it the raw text of a document. The output is a list of sentences.

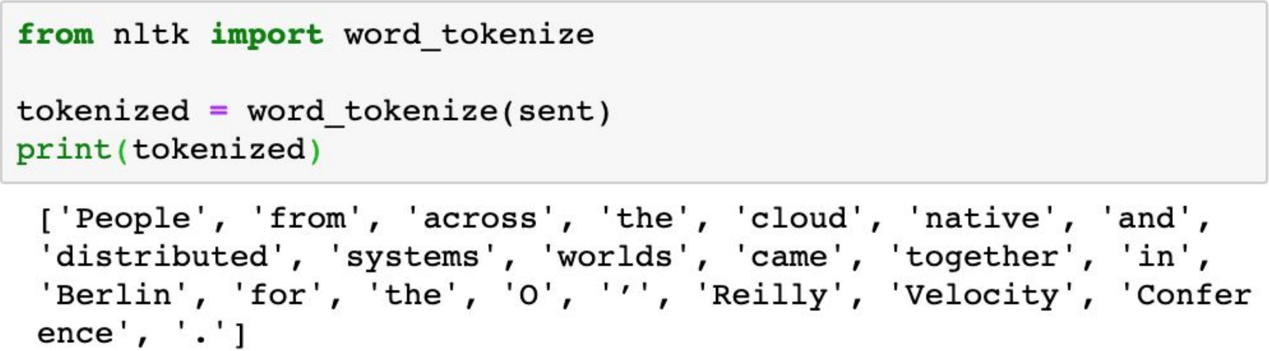


```python
from nltk import sent_tokenize

sents = sent_tokenize(doc)
```

```
['People from across the cloud native and distributed systems worlds came
together in Berlin for the O’Reilly Velocity Conference.',
 'Below you’ll find links to highlights from the event.',
 'My love letter to computer science is very short and I also forgot to mail
it James Mickens shares his concerns and frustrations with today’s
technology.',
...]
```

Word Tokenization

- Just like we deconstructed documents into sentences, we can also deconstruct sentences into tokens.
- NLTK has a `word_tokenize` function that can help us with this.
- Once we have imported it, we can just call the `word_tokenize` function and pass it a sentence. The result will be a list of tokens.

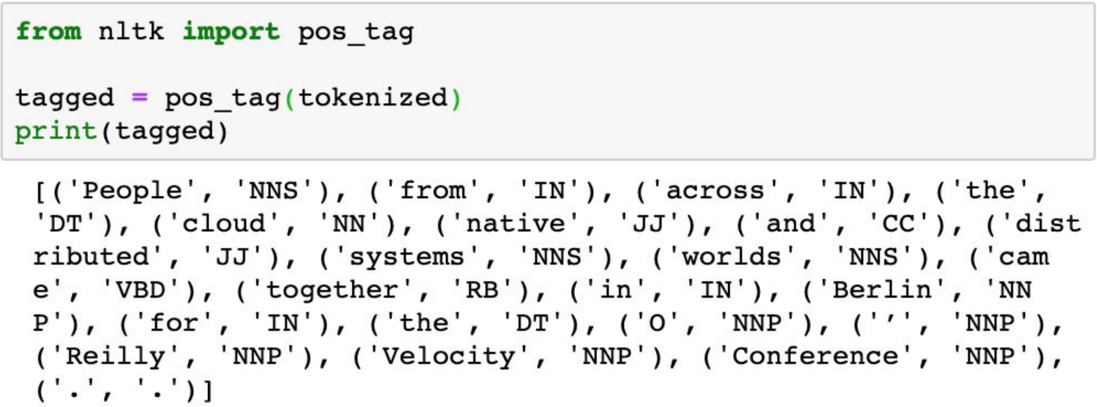


```
from nltk import word_tokenize

tokenized = word_tokenize(sent)
print(tokenized)
```

```
['People', 'from', 'across', 'the', 'cloud', 'native', 'and',
'distributed', 'systems', 'worlds', 'came', 'together', 'in',
'Berlin', 'for', 'the', 'O', '’', 'Reilly', 'Velocity', 'Confer
ence', '.']
```

Part Of Speech Tagging

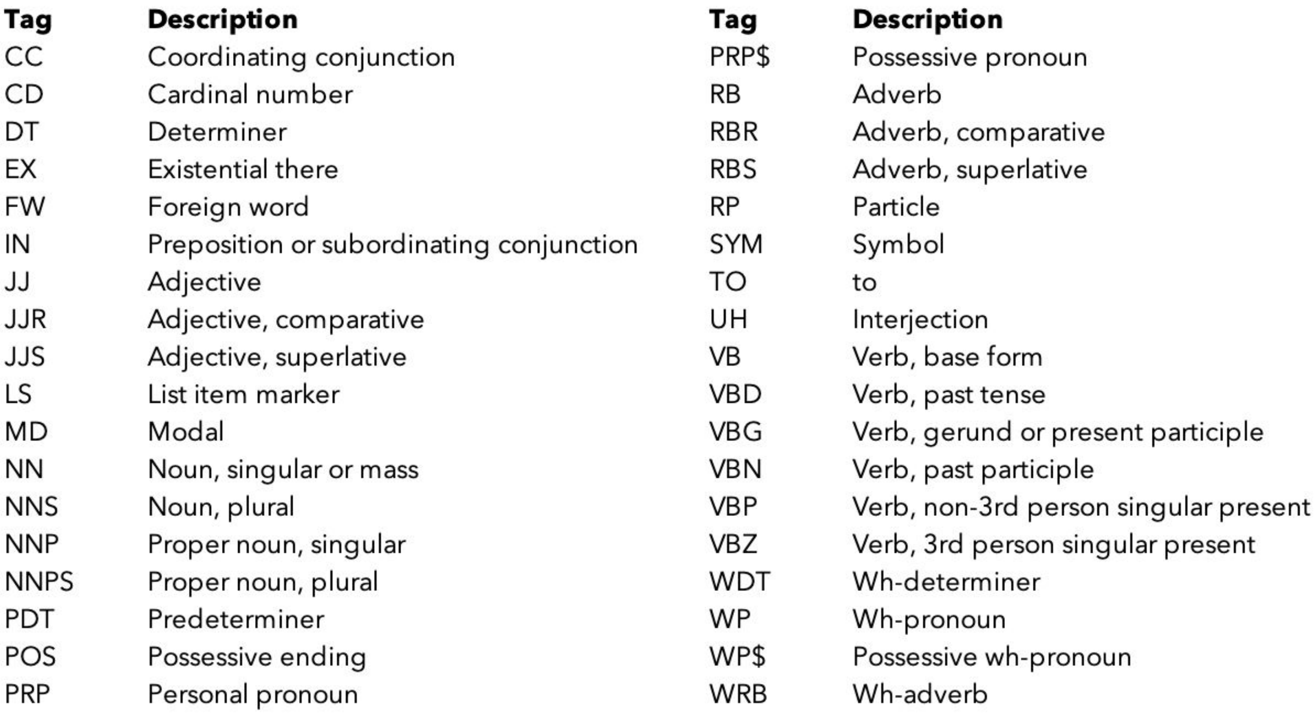
- Once we can access the tokens within sentences, we can then tag them with their part of speech (ex. noun, verb, preposition, adjective, etc.).
- Parts of speech indicate how a words function within a sentence.
- We can use NLTK’s pos_tag function to tag a tokenized sentence. The output is a list of (token, part of speech) tuples.



```
from nltk import pos_tag

tagged = pos_tag(tokenized)
print(tagged)
```

```
[('People', 'NNS'), ('from', 'IN'), ('across', 'IN'), ('the',
'DT'), ('cloud', 'NN'), ('native', 'JJ'), ('and', 'CC'), ('dist
ributed', 'JJ'), ('systems', 'NNS'), ('worlds', 'NNS'), ('cam
e', 'VBD'), ('together', 'RB'), ('in', 'IN'), ('Berlin', 'NN
P'), ('for', 'IN'), ('the', 'DT'), ('O', 'NNP'), ('’', 'NNP'),
('Reilly', 'NNP'), ('Velocity', 'NNP'), ('Conference', 'NNP'),
('.', '.')]
```

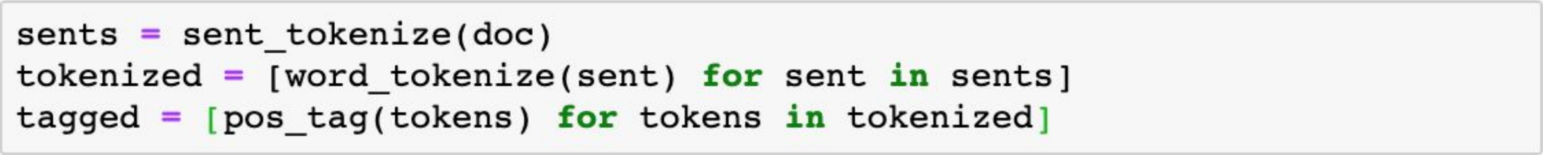
Penn Treebank Tag Set



Tag	Description
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP$	Possessive wh-pronoun
WRB	Wh-adverb

Combining Tokenization & POS Tagging

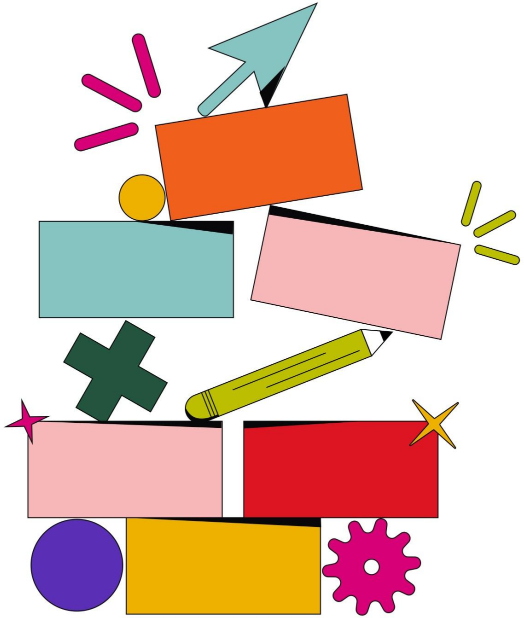
- NLTK makes it simple to perform all three of these operations in just a few lines of code.
- First, we split the document into sentences with the `sent_tokenize` function.
- Then, we iterate through each sentence and `word_tokenize` it.
- Finally, we pos_tag each list of tokens.



```
sents = sent_tokenize(doc)
tokenized = [word_tokenize(sent) for sent in sents]
tagged = [pos_tag(tokens) for tokens in tokenized]
```

Questions?

THINKFUL

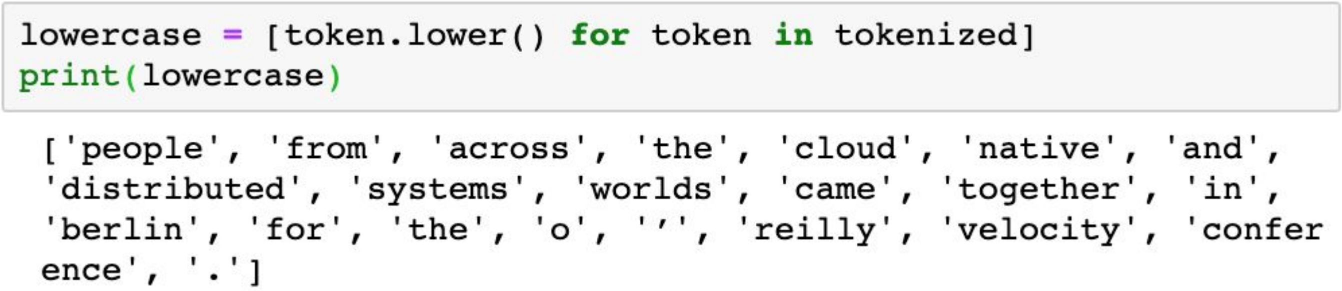


Cleaning Text Data

- Correcting typos and misspelled words.
- Dealing with abbreviations.
- Making all characters lowercase (or uppercase).
- Removing punctuation
- Removing stopwords.
- Normalizing (stemming and lemmatization).

Making Characters Lowercase

- Making all characters lowercase can help us when we are counting the number of unique words.
- When we want to filter for specific words, it relieves us from having to worry about capitalization (ex. if the word starts a sentence).
- To make a list of tokens lowercase, we can call the `lower` method.

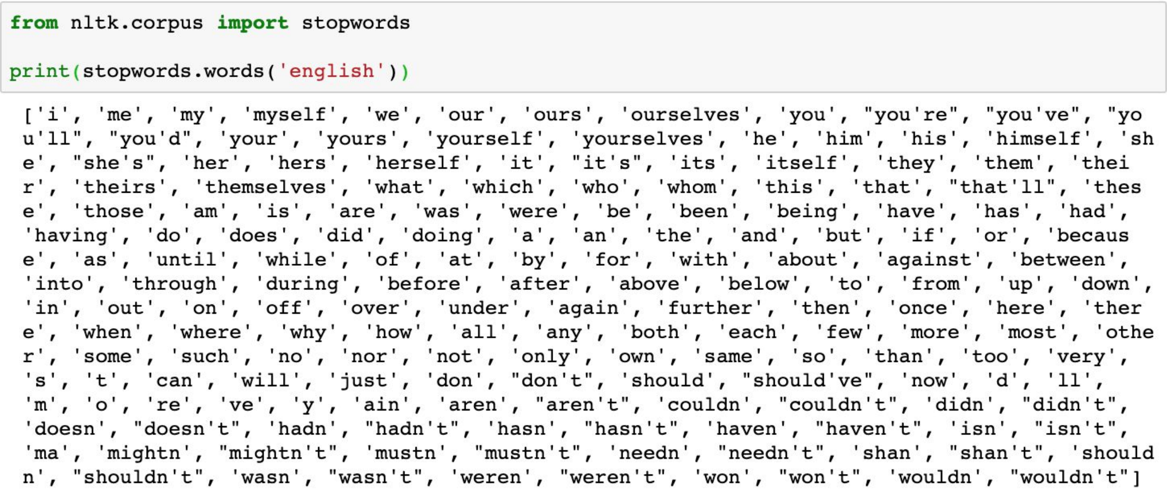


```
lowercase = [token.lower() for token in tokenized]
print(lowercase)
```

```
['people', 'from', 'across', 'the', 'cloud', 'native', 'and',
'distributed', 'systems', 'worlds', 'came', 'together', 'in',
'berlin', 'for', 'the', 'o', '’', 'reilly', 'velocity', 'confer
ence', '.']
```

Stop Word Removal

- Stop words are common words that do not add much value to whatever analysis you are going to be performing.
- NLTK has a list of built-in stop words that can be accessed as follows.

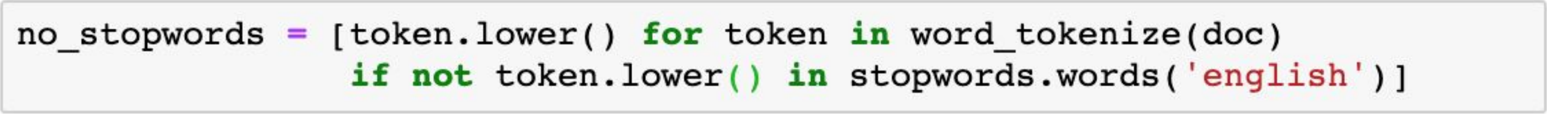


```python
from nltk.corpus import stopwords

print(stopwords.words('english'))
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "yo
u'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'sh
e', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'thei
r', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'thes
e', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'becaus
e', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between',
'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down',
'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'ther
e', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'othe
r', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very',
's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll',
'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't",
'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'should
n', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

Stop Word Removal

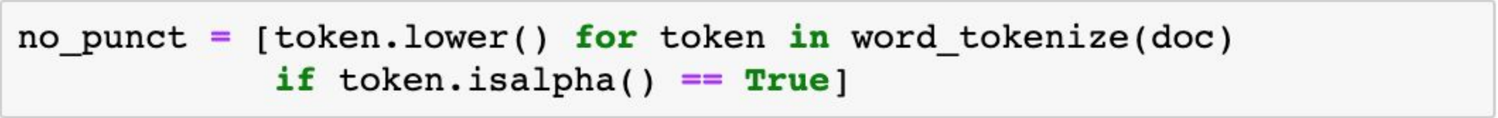
- You can remove stop words from a tokenized document using the NLTK stop words list.
- First word tokenize the document and then use a list comprehension with a conditional statement to filter out the stop words.
- You can also create your own custom stop words list if you don't want to use NLTK's stop word list.



```
no_stopwords = [token.lower() for token in word_tokenize(doc)
                if not token.lower() in stopwords.words('english')]
```

Punctuation Removal

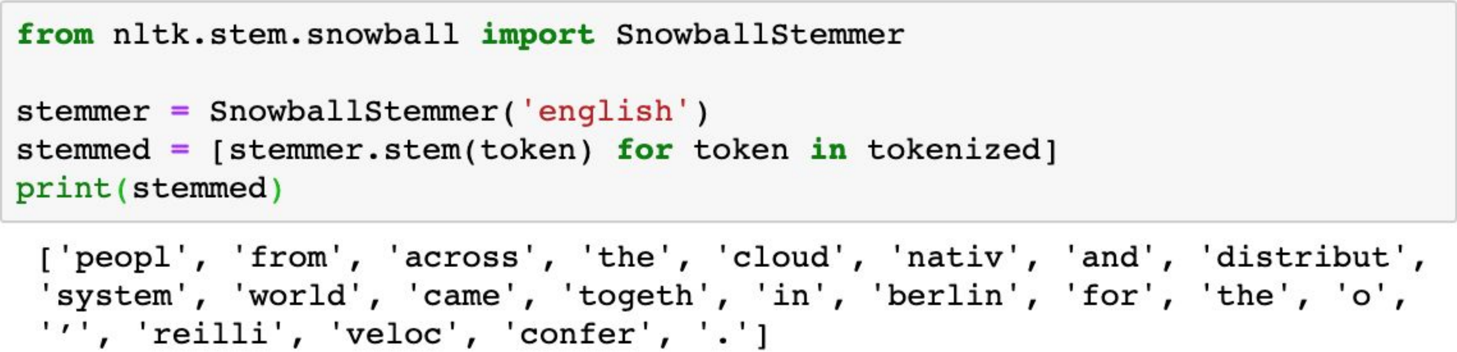
- Removing punctuation from a document is another useful text data cleaning technique.
- To do this, we would first word tokenize the document and then use a list comprehension with a conditional statement to keep only tokens that consist of alpha characters.



```
no_punct = [token.lower() for token in word_tokenize(doc)
            if token.isalpha() == True]
```

Normalization - Stemming

- NLTK comes with a few different stemmers. For most applications, the Snowball stemmer performs pretty well.
- After importing, we can call the `SnowballStemmer` function and then apply its `stem` method to each token in a list of tokens.

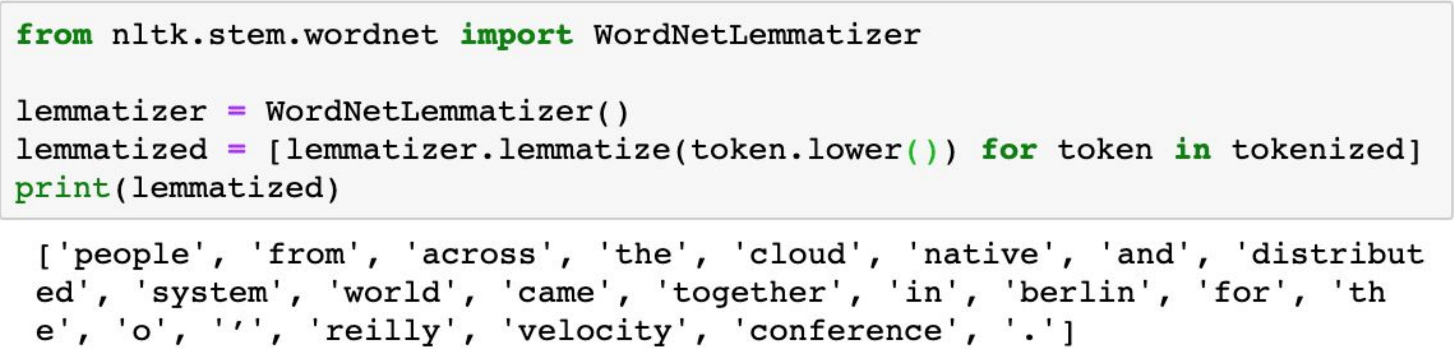


```python
from nltk.stem.snowball import SnowballStemmer

stemmer = SnowballStemmer('english')
stemmed = [stemmer.stem(token) for token in tokenized]
print(stemmed)
```

```
['peopl', 'from', 'across', 'the', 'cloud', 'nativ', 'and', 'distribut',
'system', 'world', 'came', 'togeth', 'in', 'berlin', 'for', 'the', 'o',
'’', 'reilli', 'veloc', 'confer', '.']
```

Normalization - Lemmatization

- NLTK also comes with the Wordnet Lemmatizer that strips words down to their root.
- After importing, we can call the WordNetLemmatizer function and then apply its lemmatize method to each token in a list of tokens.
- Note: Lemmatization is more computationally expensive than stemming.



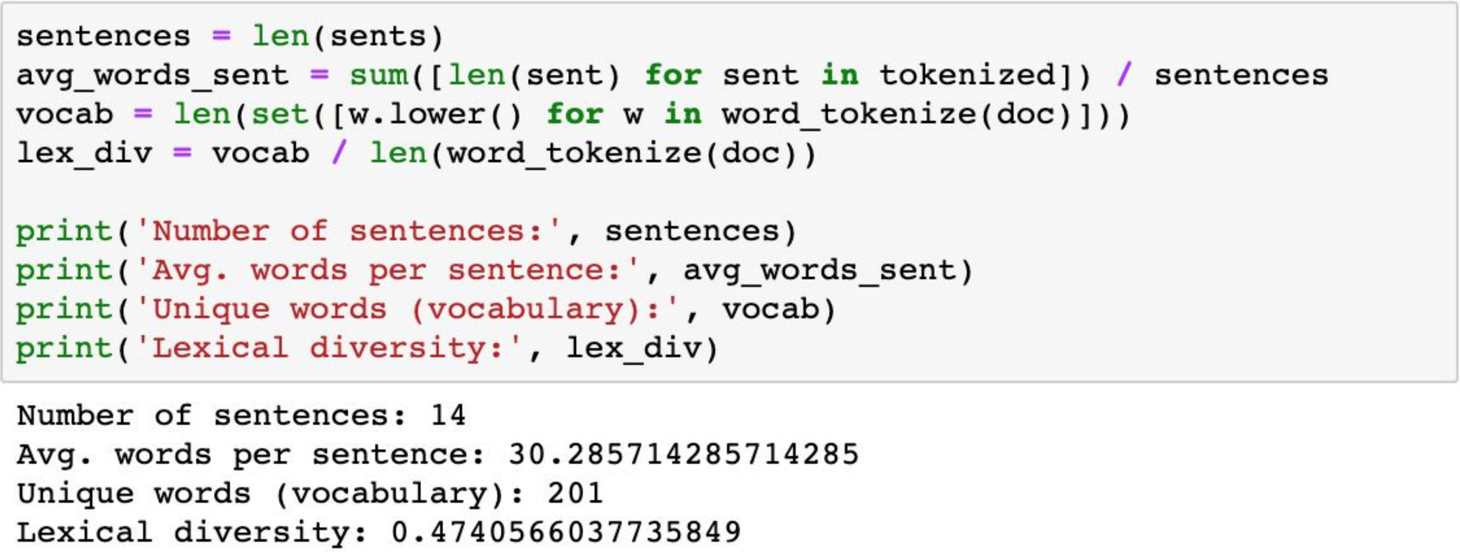
```python
from nltk.stem.wordnet import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
lemmatized = [lemmatizer.lemmatize(token.lower()) for token in tokenized]
print(lemmatized)
```

```
['people', 'from', 'across', 'the', 'cloud', 'native', 'and', 'distribut
ed', 'system', 'world', 'came', 'together', 'in', 'berlin', 'for', 'th
e', 'o', '’', 'reilly', 'velocity', 'conference', '.']
```

Document Statistics

- Now that we know how to and clean text data, we are able to compute a few different statistics at the document level.
- Number of sentences for each document.
- Average number of words per sentence for each document.
- Number of unique words (vocabulary) used in each document.
- Ratio of unique words to total words (lexical diversity) of each document.

Document Statistics

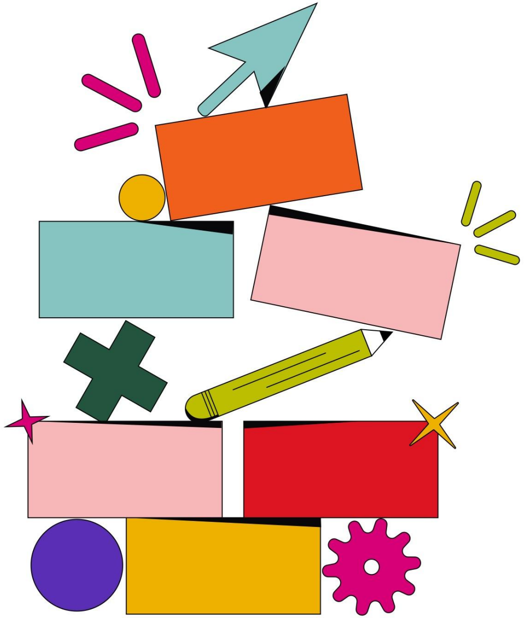


```python
sentences = len(sents)
avg_words_sent = sum([len(sent) for sent in tokenized]) / sentences
vocab = len(set([w.lower() for w in word_tokenize(doc)]))
lex_div = vocab / len(word_tokenize(doc))

print('Number of sentences:', sentences)
print('Avg. words per sentence:', avg_words_sent)
print('Unique words (vocabulary):', vocab)
print('Lexical diversity:', lex_div)
```

```
Number of sentences: 14
Avg. words per sentence: 30.285714285714285
Unique words (vocabulary): 201
Lexical diversity: 0.4740566037735849
```

Questions?

THINKFUL



Summary

Brief review, should call back to the objective and make the direct connection for how the objective has now been achieved.

- What text preprocessing is and how a document can be deconstructed.
- Sentence tokenization, word tokenization, and part of speech tagging.
- What types of activities cleaning text data entails.
- How to make characters lowercase.
- What stop words are and how to remove them from documents.
- How to normalize tokens using stemming and lemmatization.
- A few useful document-level statistics we can compute with clean, preprocessed data.

Assignment

1. See Jupyter Notebook.

THINKFUL

Thank You

Text Data Cleaning and Preprocessing

Warm Up

- What do you anticipate is involved with preprocessing and cleaning text data?

High Level Agenda

- Text Data Preprocessing
- Deconstructing Documents
- Sentence Tokenization
- Word Tokenization
- Part of Speech Tagging
- Cleaning Text Data
- Making Characters Lowercase
- Stop Word Removal
- Normalization

Text Data Preprocessing

- Before we can extract any information or perform any analytics on our text data, we need to preprocess it.

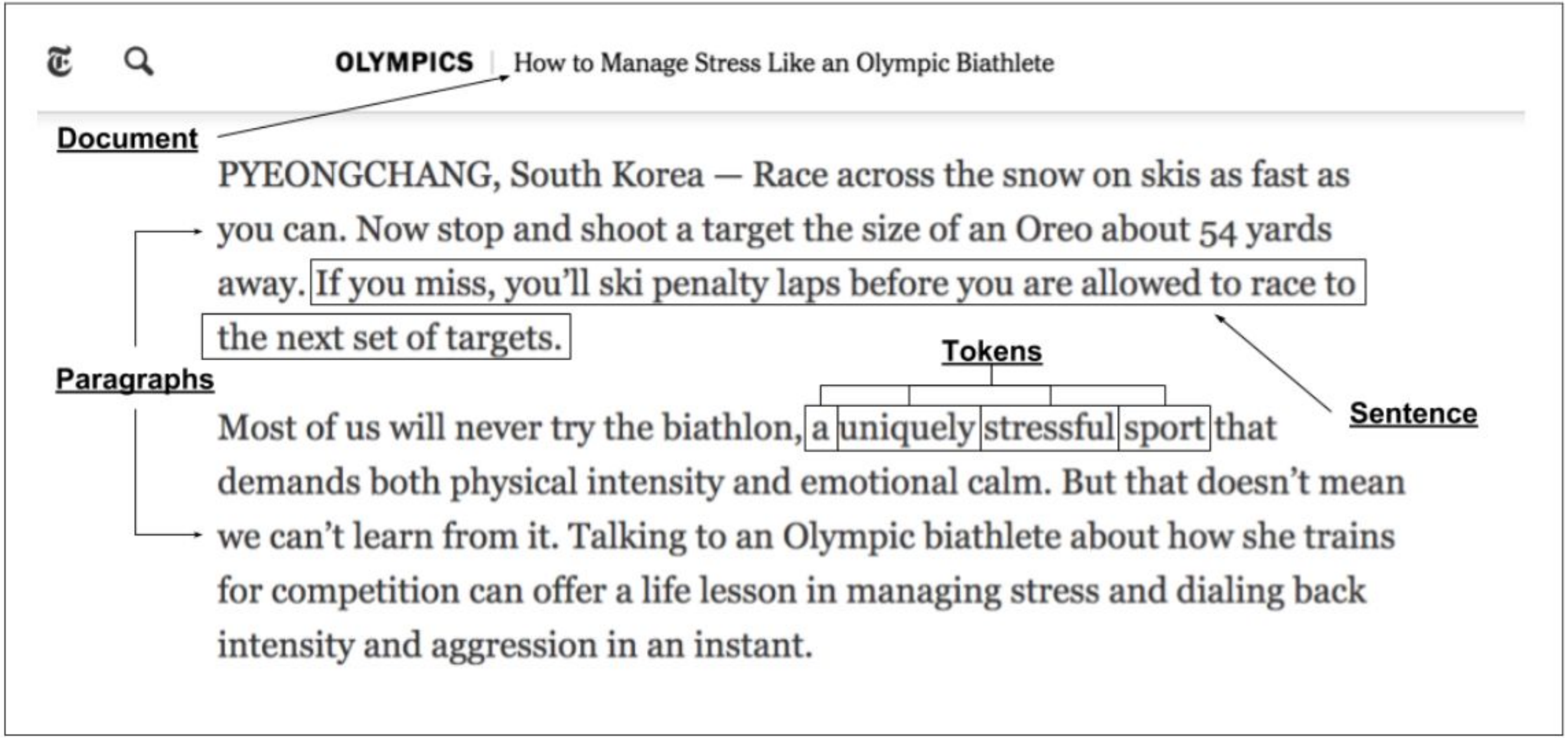
- Preprocessing will help us deconstruct our documents into their components so that we can better identify the information we are looking for and convert text into numbers that can be computed upon.

- The major tasks involved in preprocessing include tokenization, part of speech tagging, and cleaning text.

Deconstructing Documents

- The first part of preprocessing involves tokenizing our documents or deconstructing them into the components they are comprised of.
- Paragraphs - units of document structure.
- Sentences - units of discourse.
- Tokens - syntactic units of language.

Deconstructing Documents



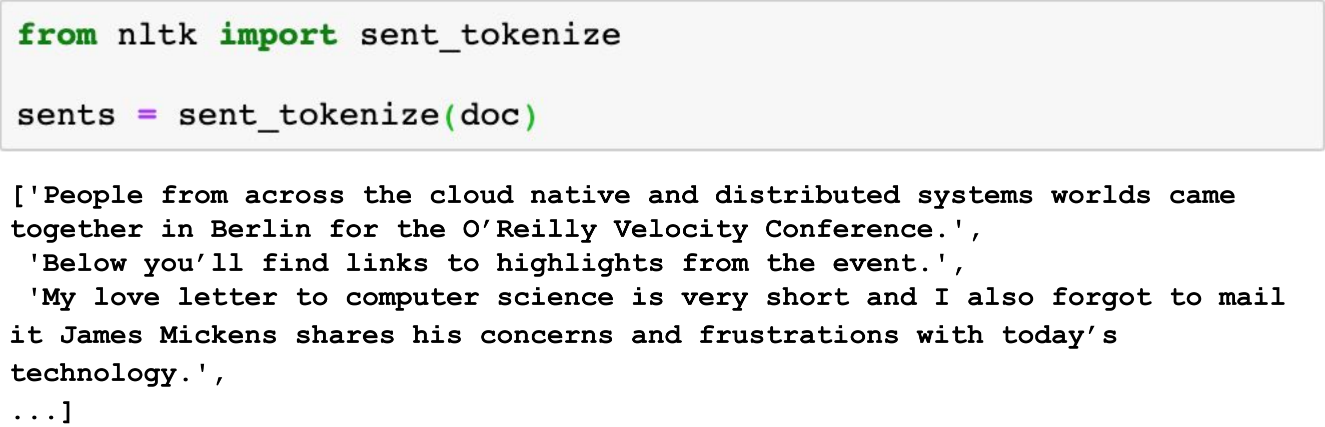
OLYMPICS | How to Manage Stress Like an Olympic Biathlete

PYEONGCHANG, South Korea — Race across the snow on skis as fast as you can. Now stop and shoot a target the size of an Oreo about 54 yards away. If you miss, you'll ski penalty laps before you are allowed to race to the next set of targets.

Most of us will never try the biathlon, a uniquely stressful sport that demands both physical intensity and emotional calm. But that doesn't mean we can't learn from it. Talking to an Olympic biathlete about how she trains for competition can offer a life lesson in managing stress and dialing back intensity and aggression in an instant.

Sentence Tokenization

- Identifying and extracting the individual sentences in a document, typically done with the use of a trained tokenizer.
- NLTK has a `sent_tokenize` function that can help us do this.
- Once we have imported it, we can just call the `sent_tokenize` function and pass it the raw text of a document. The output is a list of sentences.

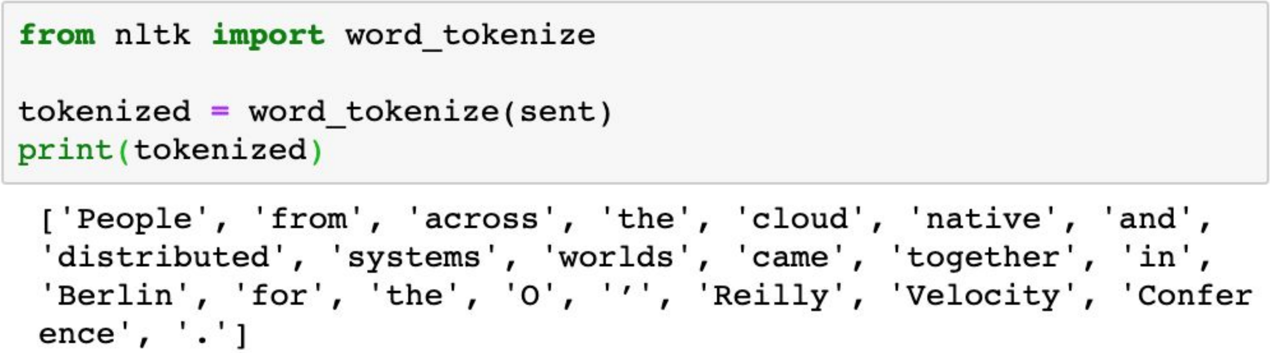


```python
from nltk import sent_tokenize

sents = sent_tokenize(doc)
```

```
['People from across the cloud native and distributed systems worlds came
together in Berlin for the O’Reilly Velocity Conference.',
 'Below you’ll find links to highlights from the event.',
 'My love letter to computer science is very short and I also forgot to mail
it James Mickens shares his concerns and frustrations with today’s
technology.',
...]
```

Word Tokenization

- Just like we deconstructed documents into sentences, we can also deconstruct sentences into tokens.
- NLTK has a `word_tokenize` function that can help us with this.
- Once we have imported it, we can just call the `word_tokenize` function and pass it a sentence. The result will be a list of tokens.

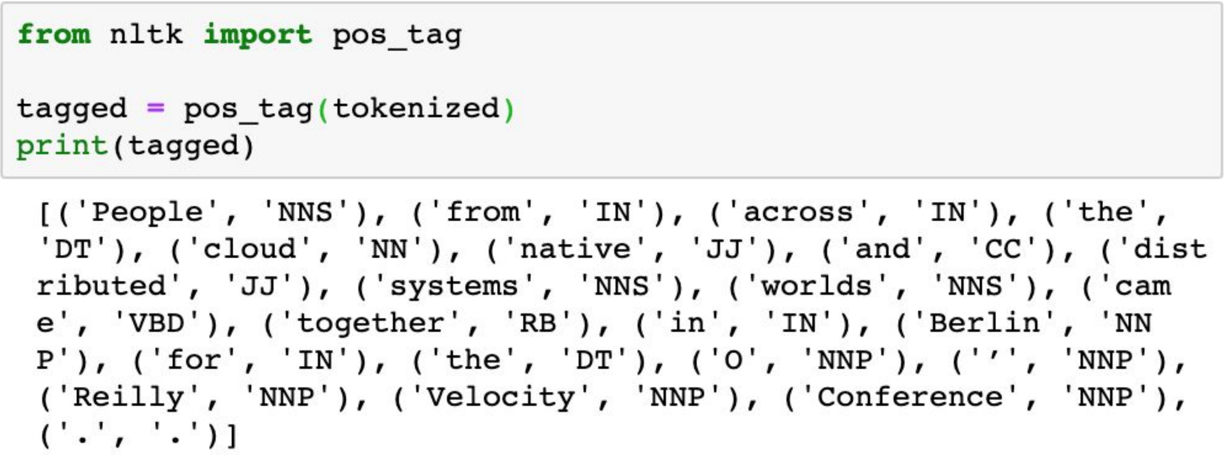


```
from nltk import word_tokenize

tokenized = word_tokenize(sent)
print(tokenized)
```

```
['People', 'from', 'across', 'the', 'cloud', 'native', 'and',
'distributed', 'systems', 'worlds', 'came', 'together', 'in',
'Berlin', 'for', 'the', 'O', '’', 'Reilly', 'Velocity', 'Confer
ence', '.']
```

Part of Speech Tagging

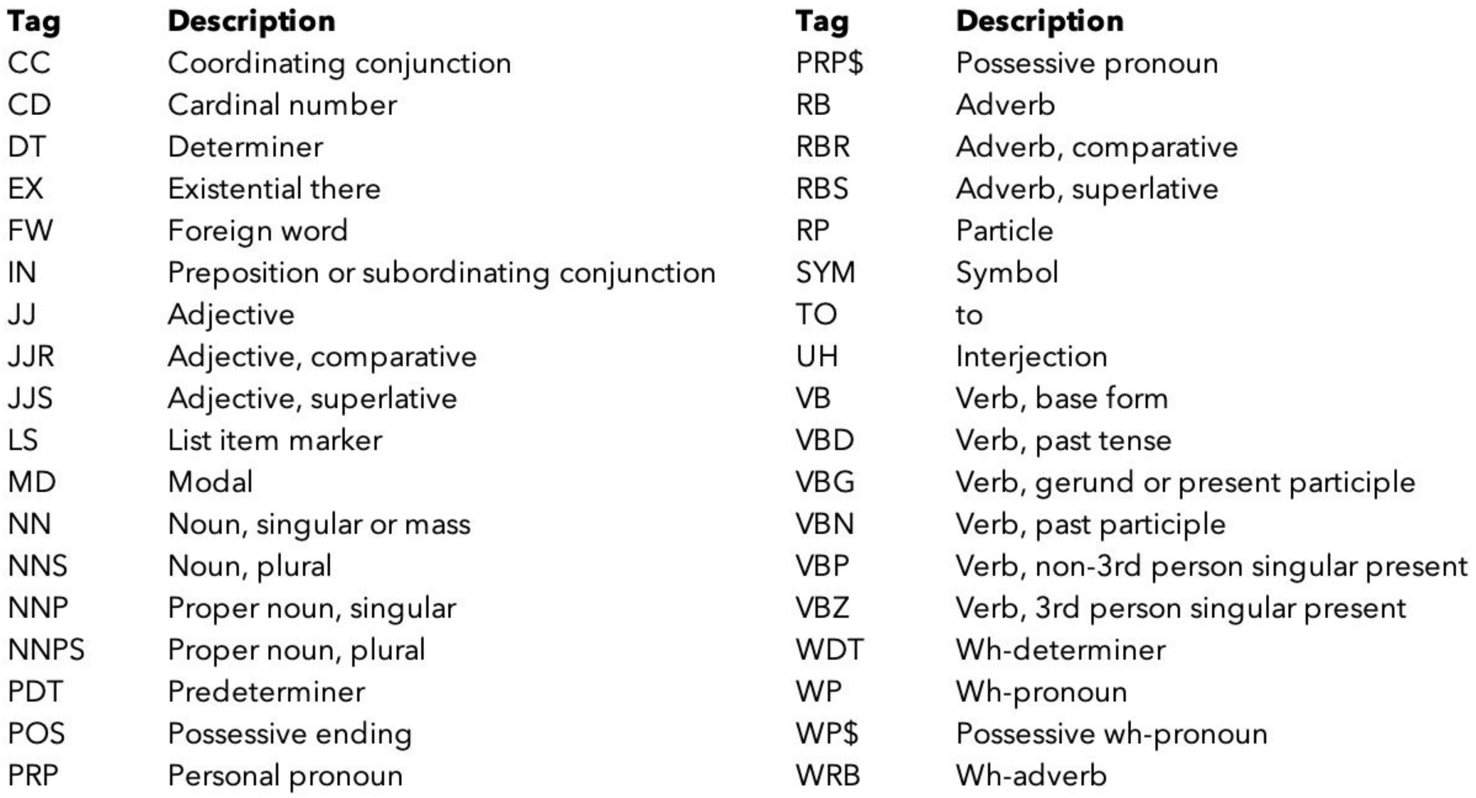
- Once we can access the tokens within sentences, we can then tag them with their part of speech (ex. noun, verb, preposition, adjective, etc.).
- Parts of speech indicate how a words function within a sentence.
- We can use NLTK’s pos_tag function to tag a tokenized sentence. The output is a list of (token, part of speech) tuples.



```
from nltk import pos_tag

tagged = pos_tag(tokenized)
print(tagged)
```

```
[('People', 'NNS'), ('from', 'IN'), ('across', 'IN'), ('the',
'DT'), ('cloud', 'NN'), ('native', 'JJ'), ('and', 'CC'), ('dist
ributed', 'JJ'), ('systems', 'NNS'), ('worlds', 'NNS'), ('cam
e', 'VBD'), ('together', 'RB'), ('in', 'IN'), ('Berlin', 'NN
P'), ('for', 'IN'), ('the', 'DT'), ('O', 'NNP'), ('’', 'NNP'),
('Reilly', 'NNP'), ('Velocity', 'NNP'), ('Conference', 'NNP'),
('.', '.')]
```

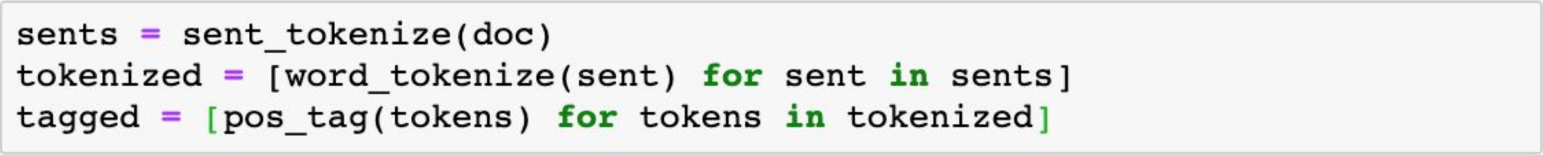
Penn Treebank Tag Set



Tag	Description	Tag	Description
CC	Coordinating conjunction	PRP$	Possessive pronoun
CD	Cardinal number	RB	Adverb
DT	Determiner	RBR	Adverb, comparative
EX	Existential there	RBS	Adverb, superlative
FW	Foreign word	RP	Particle
IN	Preposition or subordinating conjunction	SYM	Symbol
JJ	Adjective	TO	to
JJR	Adjective, comparative	UH	Interjection
JJS	Adjective, superlative	VB	Verb, base form
LS	List item marker	VBD	Verb, past tense
MD	Modal	VBG	Verb, gerund or present participle
NN	Noun, singular or mass	VBN	Verb, past participle
NNS	Noun, plural	VBP	Verb, non-3rd person singular present
NNP	Proper noun, singular	VBZ	Verb, 3rd person singular present
NNPS	Proper noun, plural	WDT	Wh-determiner
PDT	Predeterminer	WP	Wh-pronoun
POS	Possessive ending	WP$	Possessive wh-pronoun
PRP	Personal pronoun	WRB	Wh-adverb

Combining Tokenization and POS Tagging

- NLTK makes it simple to perform all three of these operations in just a few lines of code.
- First, we split the document into sentences with the `sent_tokenize` function.
- Then, we iterate through each sentence and `word_tokenize` it.
- Finally, we pos_tag each list of tokens.



```
sents = sent_tokenize(doc)
tokenized = [word_tokenize(sent) for sent in sents]
tagged = [pos_tag(tokens) for tokens in tokenized]
```

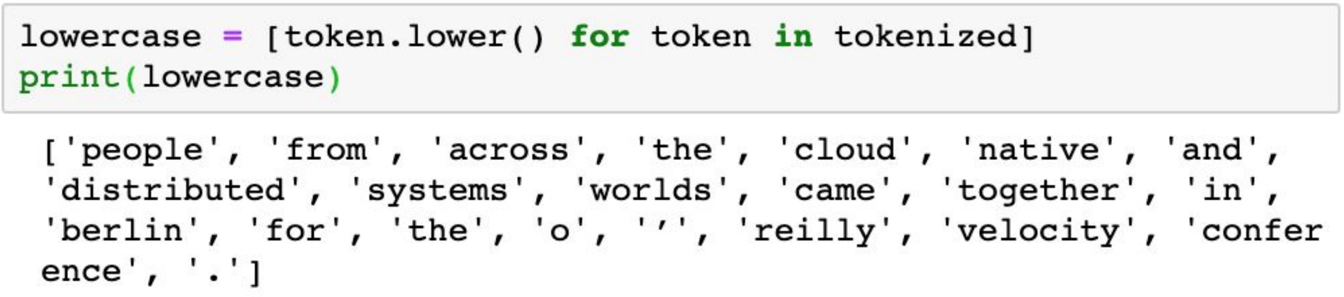
Questions?

Cleaning Text Data

- Correcting typos and misspelled words.
- Dealing with abbreviations.
- Making all characters lowercase (or uppercase).
- Removing punctuation
- Removing stopwords.
- Normalizing (stemming and lemmatization).

Making Characters Lowercase

- Making all characters lowercase can help us when we are counting the number of unique words.
- When we want to filter for specific words, it relieves us from having to worry about capitalization (ex. if the word starts a sentence).
- To make a list of tokens lowercase, we can call the `lower` method.

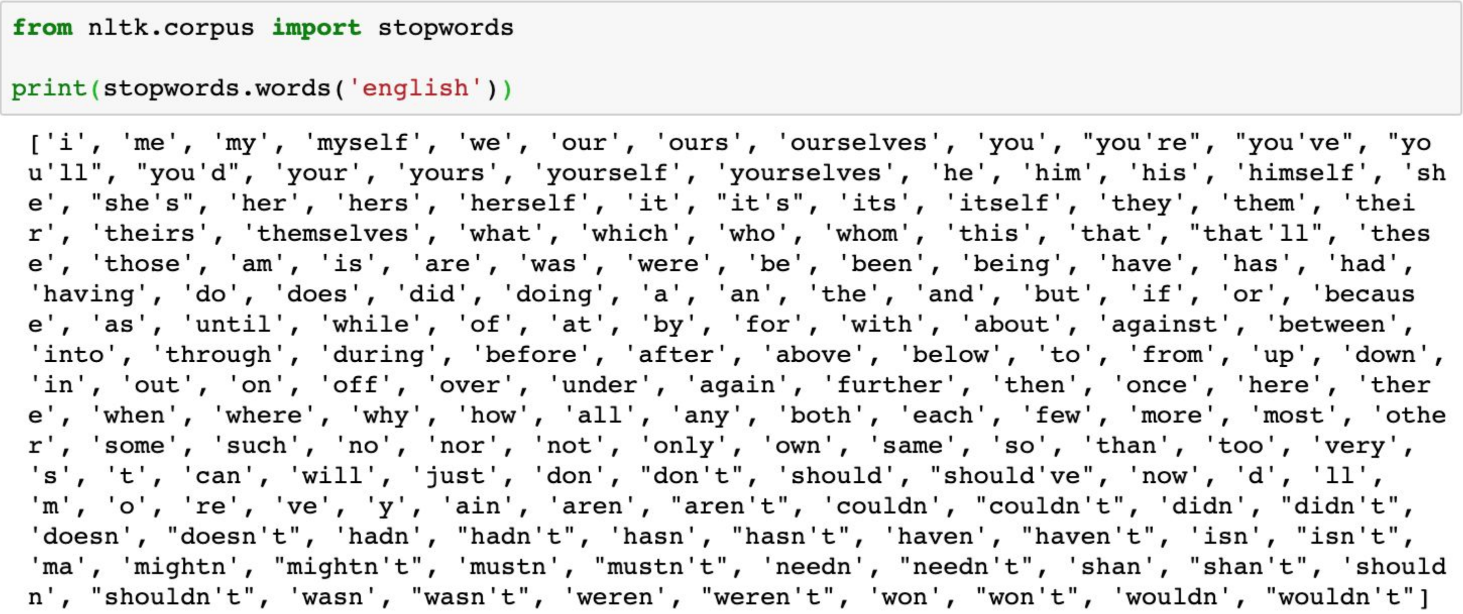


```
lowercase = [token.lower() for token in tokenized]
print(lowercase)
```

```
['people', 'from', 'across', 'the', 'cloud', 'native', 'and',
'distributed', 'systems', 'worlds', 'came', 'together', 'in',
'berlin', 'for', 'the', 'o', '’', 'reilly', 'velocity', 'confer
ence', '.']
```

Stop Word Removal

- Stop words are common words that do not add much value to whatever analysis you are going to be performing.
- NLTK has a list of built-in stop words that can be accessed as follows.

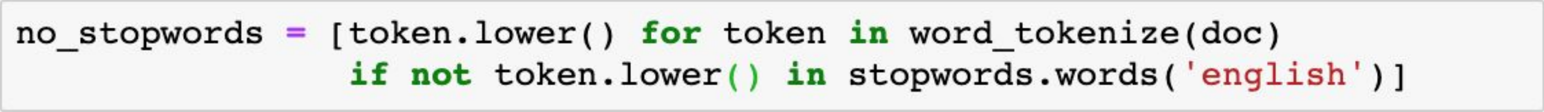


```python
from nltk.corpus import stopwords

print(stopwords.words('english'))
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "yo
u'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'sh
e', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'thei
r', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'thes
e', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'becaus
e', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between',
'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down',
'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'ther
e', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'othe
r', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very',
's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll',
'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't",
'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'should
n', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

Stop Word Removal

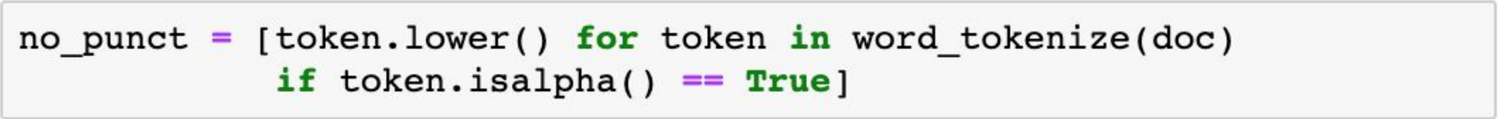
- You can remove stop words from a tokenized document using the NLTK stop words list.
- First word tokenize the document and then use a list comprehension with a conditional statement to filter out the stop words.
- You can also create your own custom stop words list if you don’t want to use NLTK’s stop word list.



```
no_stopwords = [token.lower() for token in word_tokenize(doc)
                if not token.lower() in stopwords.words('english')]
```

Punctuation Removal

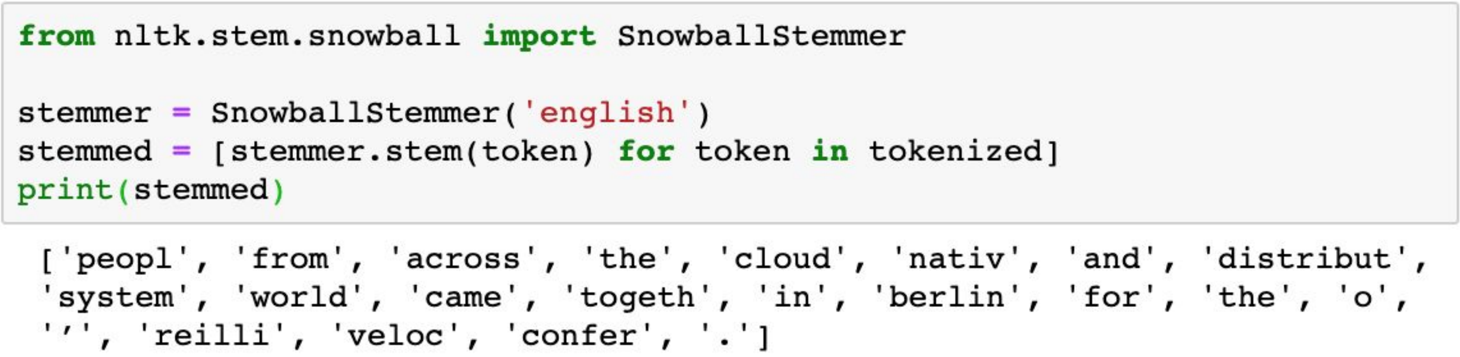
- Removing punctuation from a document is another useful text data cleaning technique.
- To do this, we would first word tokenize the document and then use a list comprehension with a conditional statement to keep only tokens that consist of alpha characters.



```
no_punct = [token.lower() for token in word_tokenize(doc)
            if token.isalpha() == True]
```

Normalization - Stemming

- NLTK comes with a few different stemmers. For most applications, the Snowball stemmer performs pretty well.
- After importing, we can call the `SnowballStemmer` function and then apply its `stem` method to each token in a list of tokens.

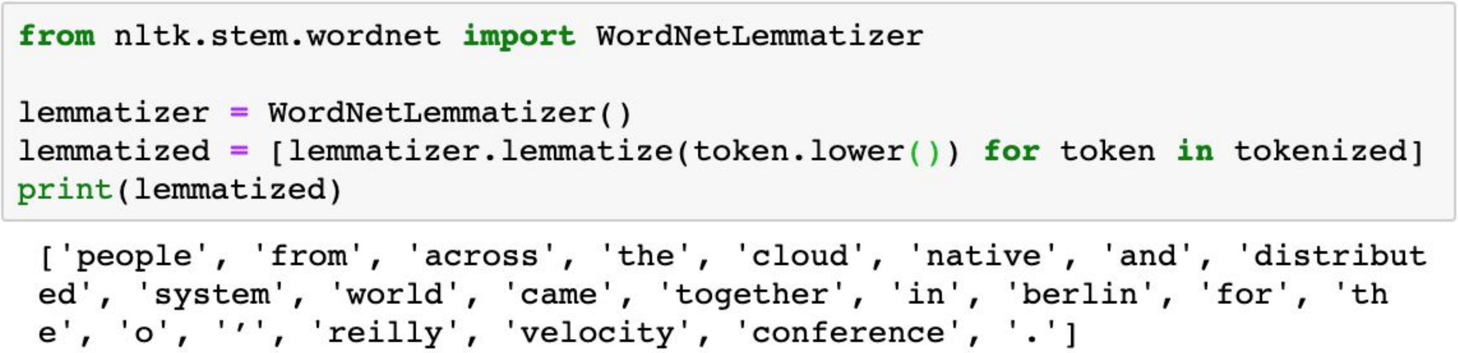


```python
from nltk.stem.snowball import SnowballStemmer

stemmer = SnowballStemmer('english')
stemmed = [stemmer.stem(token) for token in tokenized]
print(stemmed)
```

```
['peopl', 'from', 'across', 'the', 'cloud', 'nativ', 'and', 'distribut',
'system', 'world', 'came', 'togeth', 'in', 'berlin', 'for', 'the', 'o',
'’', 'reilli', 'veloc', 'confer', '.']
```

Normalization - Lemmatization

- NLTK also comes with the Wordnet Lemmatizer that strips words down to their root.
- After importing, we can call the `WordNetLemmatizer` function and then apply its `lemmatize` method to each token in a list of tokens.
- Note: Lemmatization is more computationally expensive than stemming.



```
from nltk.stem.wordnet import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
lemmatized = [lemmatizer.lemmatize(token.lower()) for token in tokenized]
print(lemmatized)
```

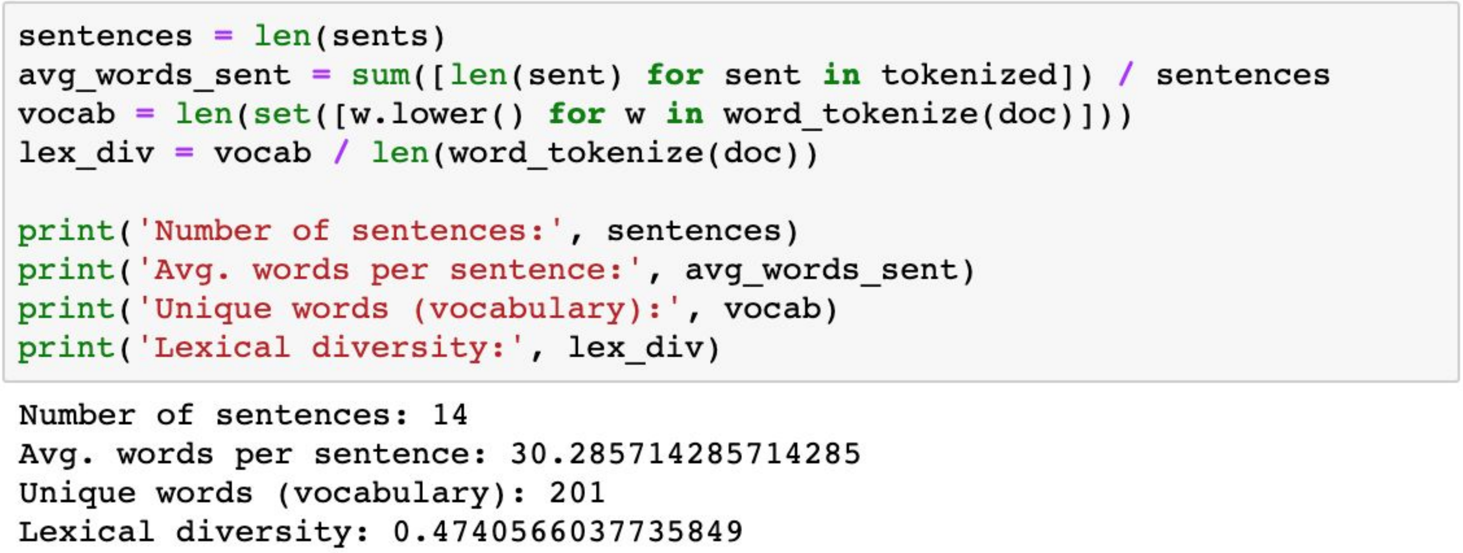
```
['people', 'from', 'across', 'the', 'cloud', 'native', 'and', 'distribut
ed', 'system', 'world', 'came', 'together', 'in', 'berlin', 'for', 'th
e', 'o', '’', 'reilly', 'velocity', 'conference', '.']
```

Document Statistics

- Now that we know how to and clean text data, we are able to compute a few different statistics at the document level.

- Number of sentences for each document.
- Average number of words per sentence for each document.
- Number of unique words (vocabulary) used in each document.
- Ratio of unique words to total words (lexical diversity) of each document.

Document Statistics



```
sentences = len(sents)
avg_words_sent = sum([len(sent) for sent in tokenized]) / sentences
vocab = len(set([w.lower() for w in word_tokenize(doc)]))
lex_div = vocab / len(word_tokenize(doc))

print('Number of sentences:', sentences)
print('Avg. words per sentence:', avg_words_sent)
print('Unique words (vocabulary):', vocab)
print('Lexical diversity:', lex_div)
```

```
Number of sentences: 14
Avg. words per sentence: 30.285714285714285
Unique words (vocabulary): 201
Lexical diversity: 0.4740566037735849
```

Questions?

Recap

In this session, we covered:

- What text preprocessing is and how a document can be deconstructed.
- Sentence tokenization, word tokenization, and part of speech tagging.
- What types of activities cleaning text data entails.
- How to make characters lowercase.
- What stop words are and how to remove them from documents.
- How to normalize tokens using stemming and lemmatization.
- A few useful document-level statistics we can compute with clean, preprocessed data.

Assignment

- See Jupyter Notebook.