ENSF 612
Lecture - Natural Language Processing (NLP)
NLP Overview – Conditional and Tokenization

Gias Uddin, Assistant Professor

Department of Electrical and Computer Engineering,

University of Calgary

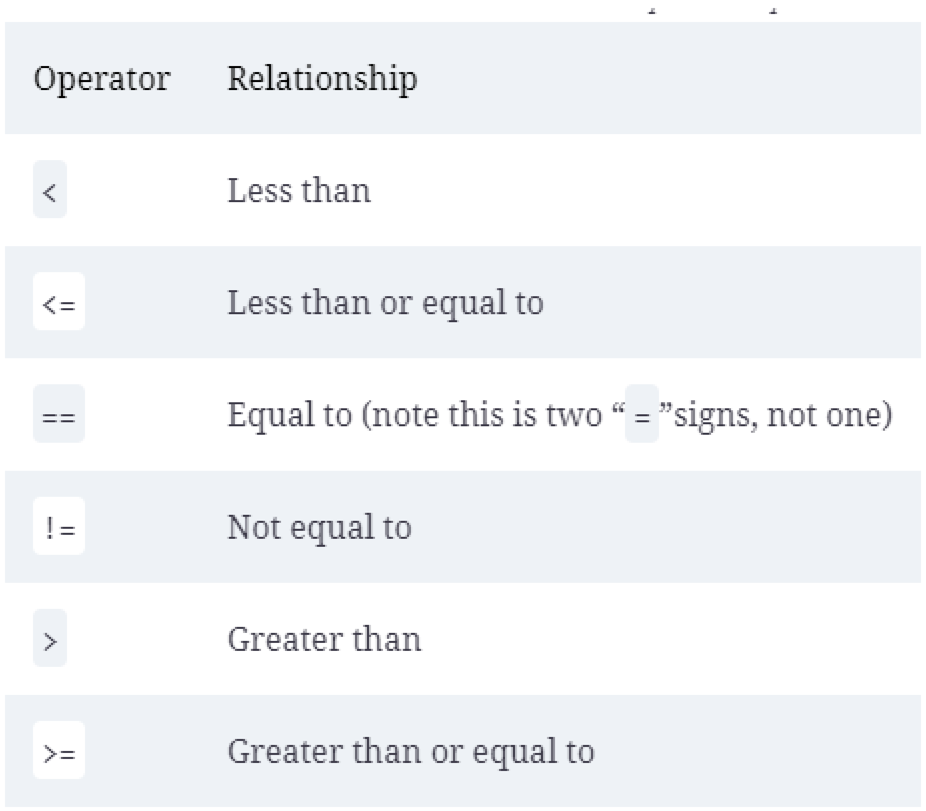
https://giasuddin.ca/

Topics

- Python Conditionals in Text Processing
- Text Preprocessing

Conditionals

Python conditionals



Operator	Relationship
`<`	Less than
`<=`	Less than or equal to
`==`	Equal to (note this is two “ `=` ”signs, not one)
`!=`	Not equal to
`>`	Greater than
`>=`	Greater than or equal to

We can apply Python conditionals on textual contents to filter the textual contents or to check for specific information. Let’s see how we can do that!

Python conditionals used in Text Processing

```python
# conditionals
sent = ['Pierre', 'Vinken', ',', '61', 'years', 'old', ',', 'will', 'join', 'the',
'board', 'as', 'a', 'nonexecutive', 'director', 'Nov.', '29', '.']
```

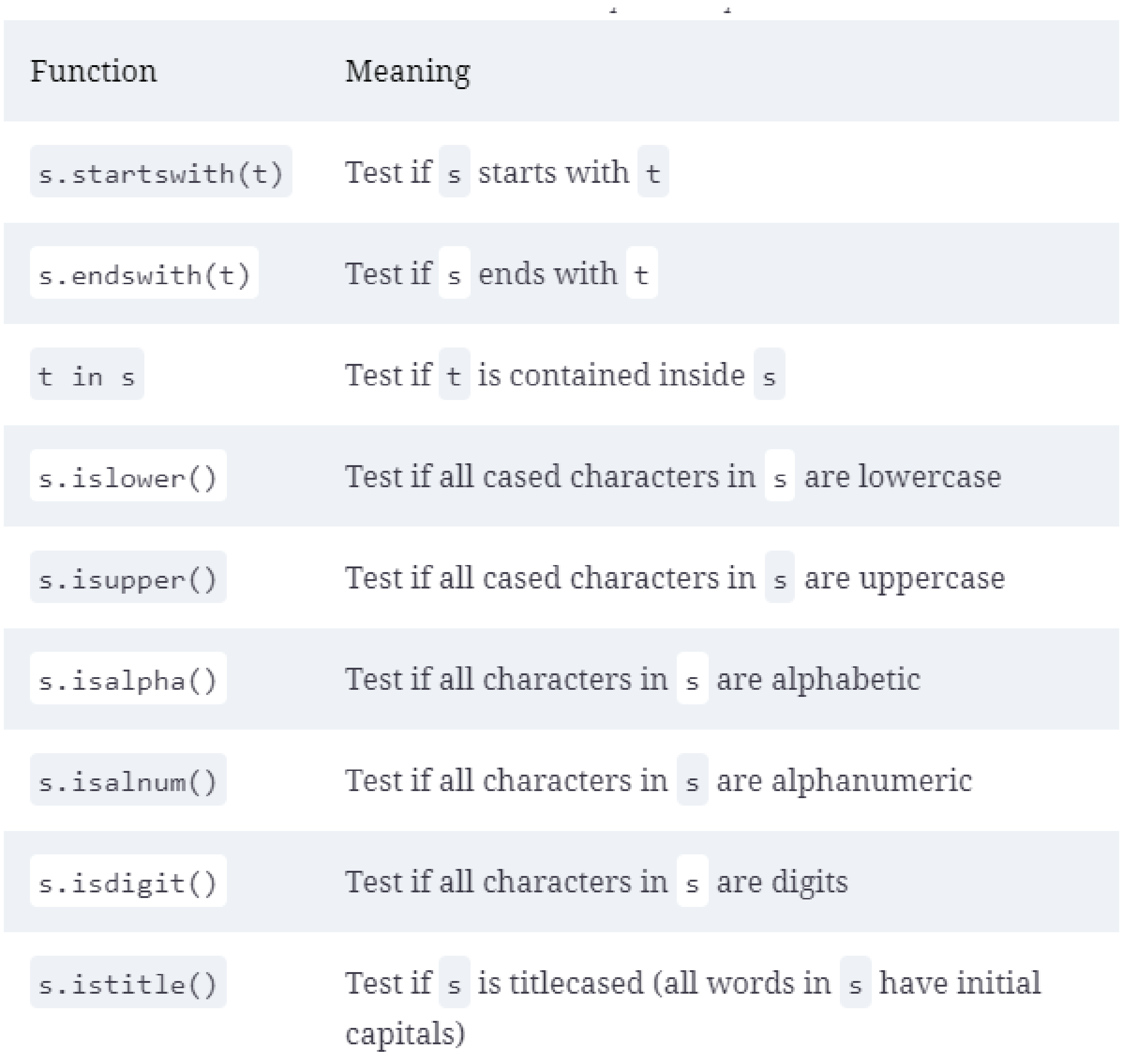
```python
# only show words with length greater then 3
for word in sent:
  if len(word) > 3:
    print(word)
```

```
Pierre
Vinken
years
will
join
board
nonexecutive
director
Nov.
```

```python
# only show words with with length equal to 4
for word in sent:
  if len(word) == 4:
    print(word)
```

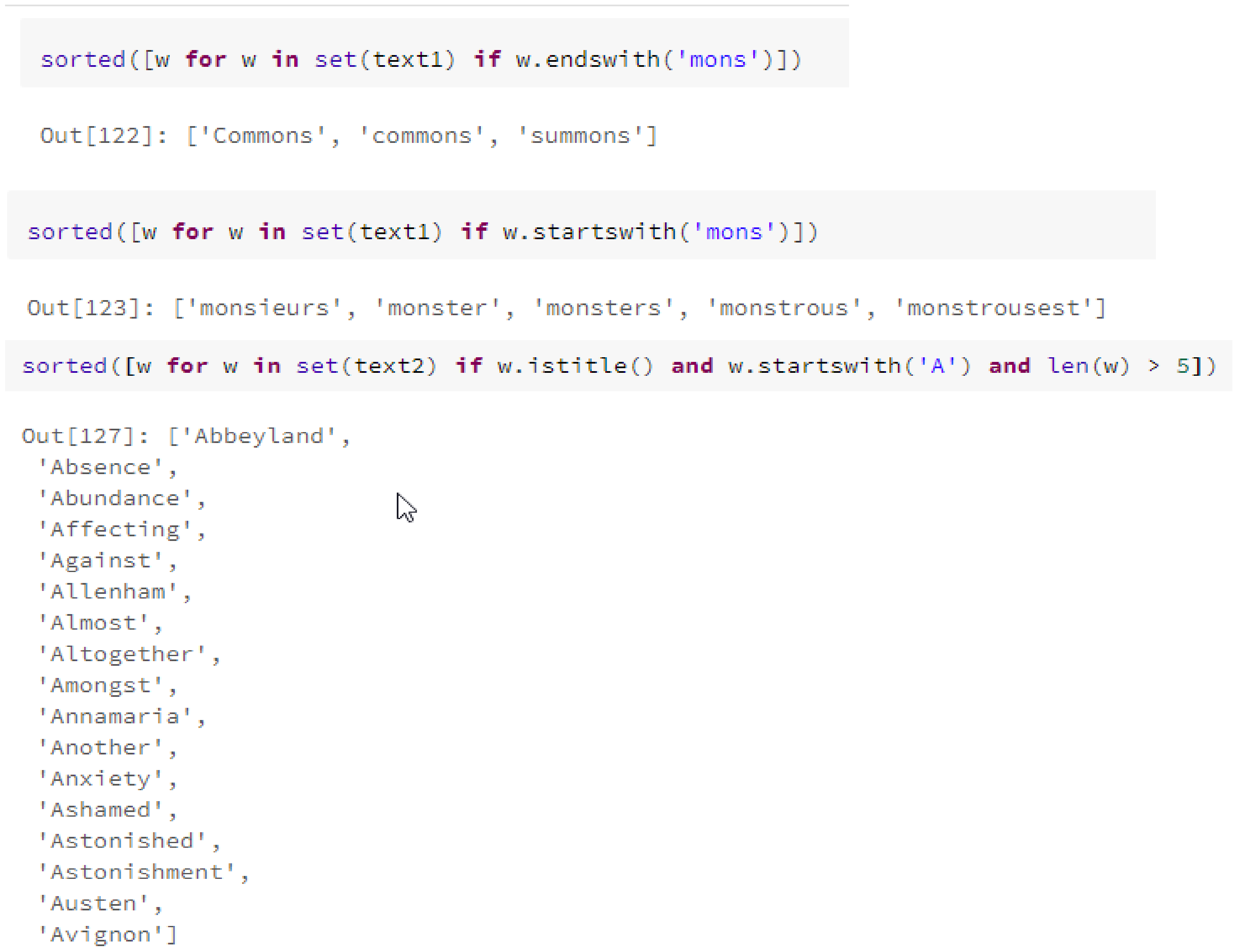
```
will
join
Nov.
```

Word specific conditionals in Python



Function	Meaning
`s.startswith(t)`	Test if `s` starts with `t`
`s.endswith(t)`	Test if `s` ends with `t`
`t in s`	Test if `t` is contained inside `s`
`s.islower()`	Test if all cased characters in `s` are lowercase
`s.isupper()`	Test if all cased characters in `s` are uppercase
`s.isalpha()`	Test if all characters in `s` are alphabetic
`s.isalnum()`	Test if all characters in `s` are alphanumeric
`s.isdigit()`	Test if all characters in `s` are digits
`s.istitle()`	Test if `s` is titlecased (all words in `s` have initial capitals)

Word specific conditionals in Python



```
sorted([w for w in set(text1) if w.endswith('mons')])
```

```
Out[122]: ['Commons', 'commons', 'summons']
```

```
sorted([w for w in set(text1) if w.startswith('mons')])
```

```
Out[123]: ['monsieurs', 'monster', 'monsters', 'monstrous', 'monstrousest']
```

```
sorted([w for w in set(text2) if w.istitle() and w.startswith('A') and len(w) > 5])
```

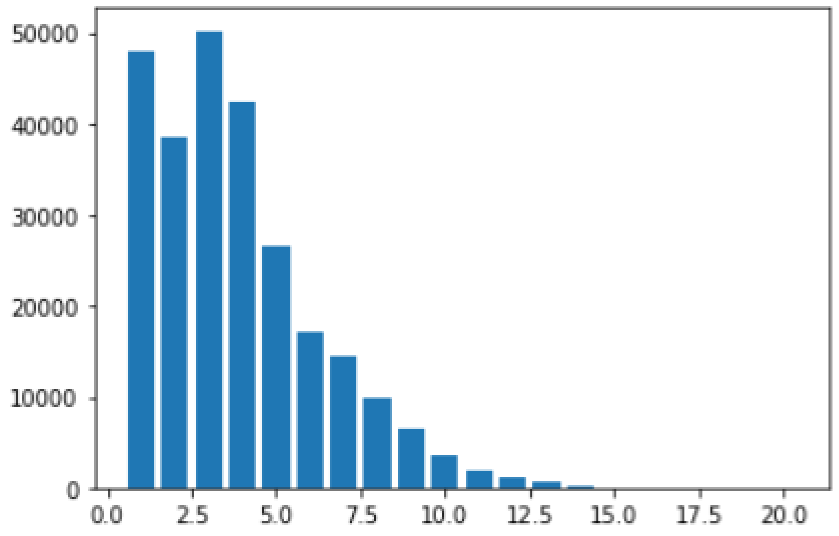
```
Out[127]: ['Abbeyland',
 'Absence',
 'Abundance',
 'Affecting',
 'Against',
 'Allenham',
 'Almost',
 'Altogether',
 'Amongst',
 'Annamaria',
 'Another',
 'Anxiety',
 'Ashamed',
 'Astonished',
 'Astonishment',
 'Austen',
 'Avignon']
```

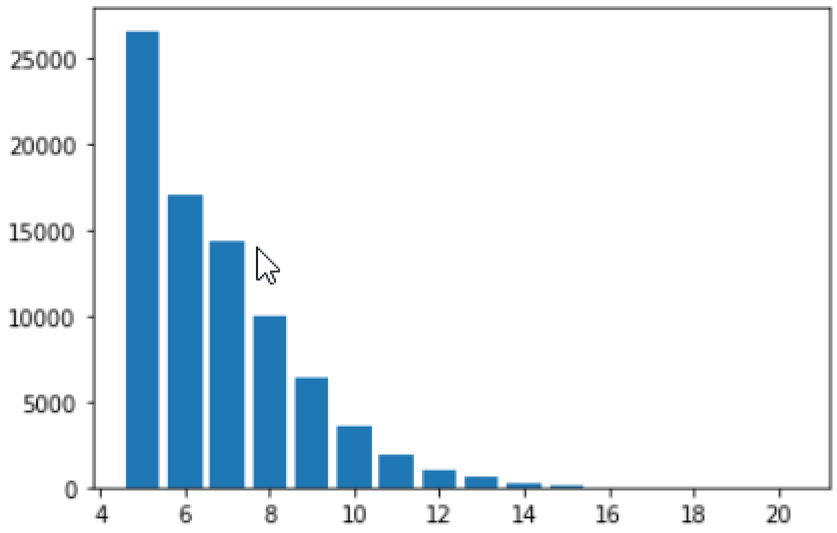
Mixing conditionals with NLTK functions

```
lens = [len(w) for w in text1]
fdist_top = FreqDist([l for l in lens if l > 4])
fdist_all = FreqDist(lens)
fdist_bottom = FreqDist([l for l in lens if l <= 4])
```

```
import matplotlib.pyplot as plt
plt.bar(*zip(*fdist_all.items()))
plt.show()
```

```
plt.bar(*zip(*fdist_top.items()))
plt.show()
```

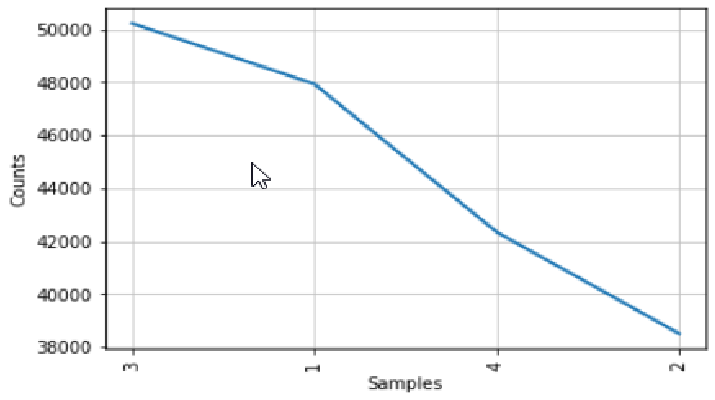




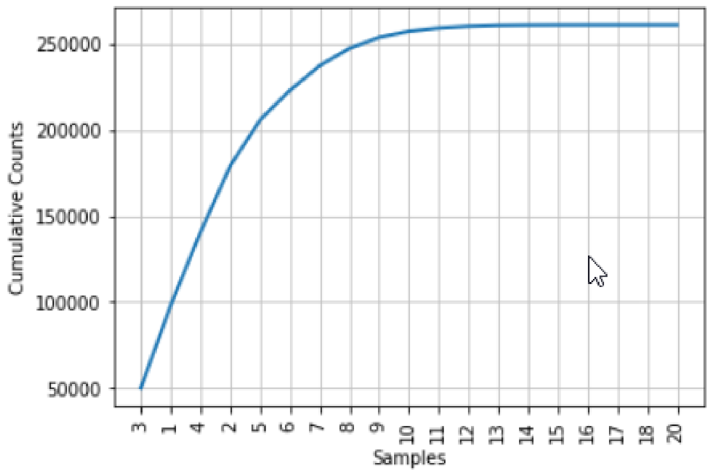
Frequency Distribution Functions in NLTK

`fdist = FreqDist(samples)`	Create a frequency distribution containing the given samples
`fdist.inc(sample)`	Increment the count for this sample
`fdist['monstrous']`	Count of the number of times a given sample occurred
`fdist.freq('monstrous')`	Frequency of a given sample
`fdist.N()`	Total number of samples
`fdist.keys()`	The samples sorted in order of decreasing frequency
`for sample in fdist:`	Iterate over the samples, in order of decreasing frequency
`fdist.max()`	Sample with the greatest count
`fdist.tabulate()`	Tabulate the frequency distribution
`fdist.plot()`	Graphical plot of the frequency distribution
`fdist.plot(cumulative=True)`	Cumulative plot of the frequency distribution
`fdist1 < fdist2`	Test if samples in `fdist1` occur less frequently than in `fdist2`

```
fdist_bottom.plot()
```



```
fdist_all.plot(cumulative=True)
```



Preprocessing of Textual Contents

1. Tokenization
   - Sentence detection
   - Word detection
2. Filter noise
   - Remove not important words/symbols
   - Fix typos
3. Tag parts of speech

Sentence Detection

Wikipedia: https://en.wikipedia.org/wiki/Sentence_boundary_disambiguation

Sentence boundary disambiguation (**SBD**), also known as **sentence breaking**, **sentence boundary detection**, and **sentence segmentation**, is the problem in natural language processing of deciding where sentences begin and end. Natural language processing tools often require their input to be divided into sentences; however, sentence boundary identification can be challenging due to the potential ambiguity of punctuation marks. In written English, a period may indicate the end of a sentence, or may denote an abbreviation, a decimal point, an ellipsis, or an email address, among other possibilities. About 47% of the periods in the Wall Street Journal corpus denote abbreviations.[1] Question marks and exclamation marks can be similarly ambiguous due to use in emoticons, computer code, and slang.

Languages like Japanese and Chinese have unambiguous sentence-ending markers.

Sentence Detection

Basic rules (from Wikipedia)

1. If it’s a period, it ends a sentence:
   - **Working:** This is sentence1. This is sentence2.
   - **Not working:** There are 59 B.Sc. students in this course, no Ph.D. students.

2. If the preceding token is in the hand-compiled list of abbreviations, then it does not end a sentence. Suppose “B.Sc.” was included in our list, but not “Ph.D.”
   - Working up to : “There are 59 B.Sc. students in this course, no Ph.”

3. If the next token is capitalized, then it ends a sentence
   - **Working** up to : “There are 59 B”

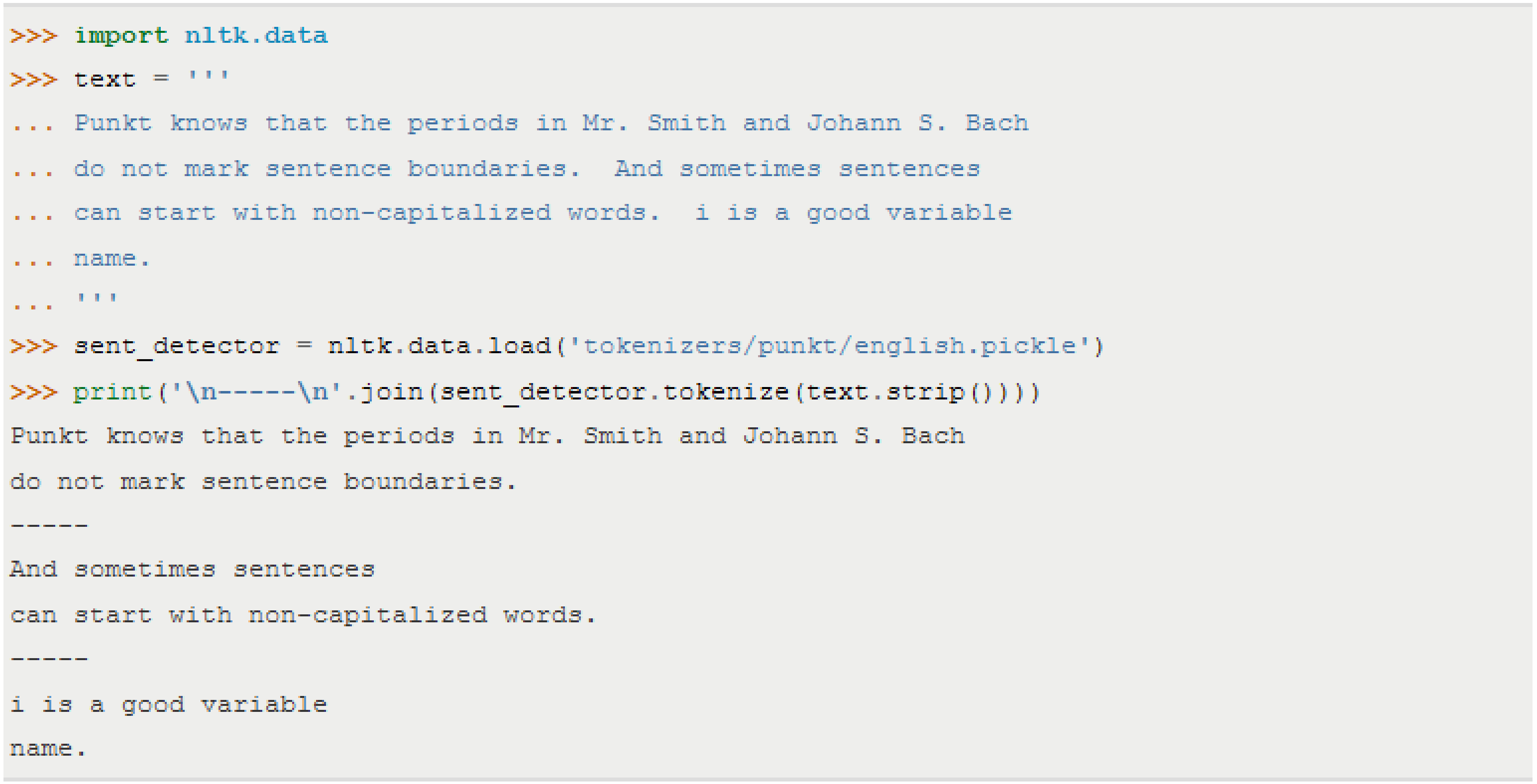
Machine learning techniques perform much better to detect sentences due to the shortcomings of rule-based techniques like above.

Sentence Detection

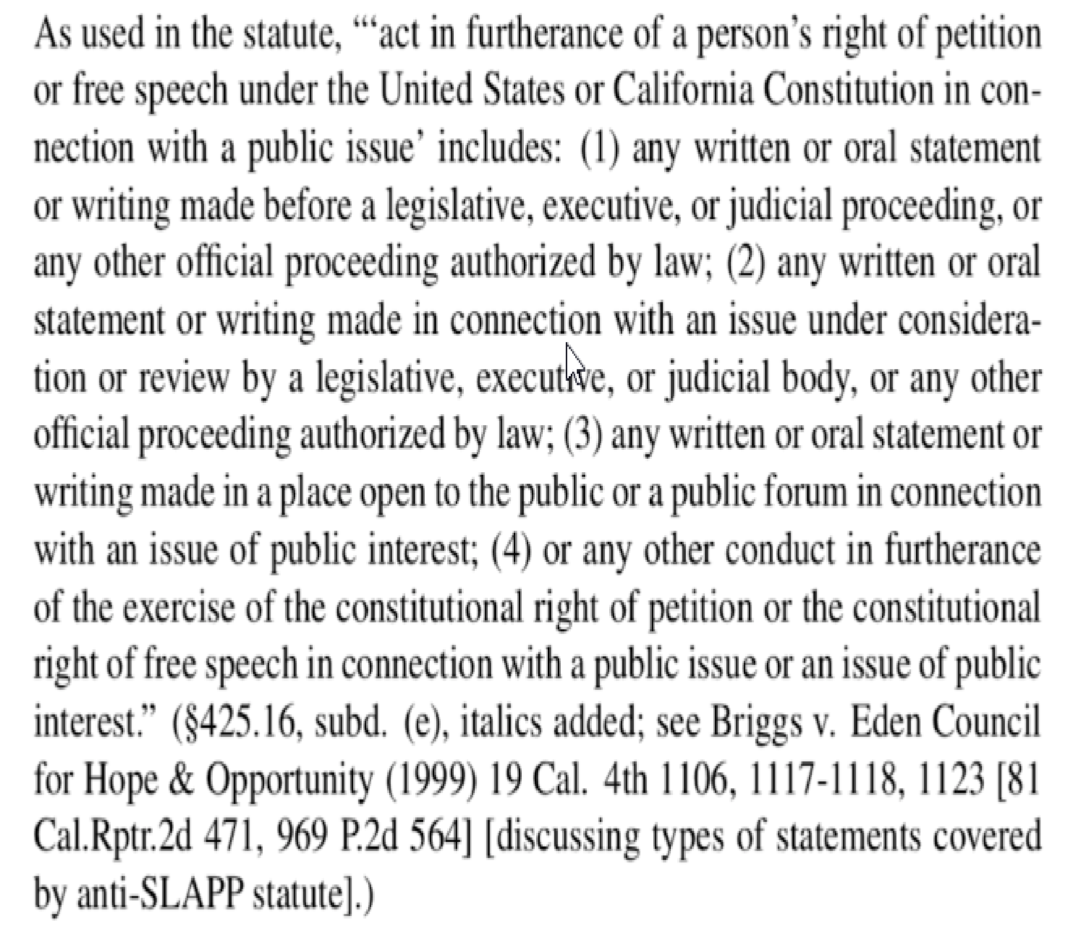
Machine Learning Models

1. NLTK Punkt Sentence Tokenizer: https://www.nltk.org/api/nltk.tokenize.html

"This tokenizer divides a text into a list of sentences by using an unsupervised algorithm to build a model for abbreviation words, collocations, and words that start sentences. It must be trained on a large collection of plaintext in the target language before it can be used."



```
>>> import nltk.data
>>> text = '''
... Punkt knows that the periods in Mr. Smith and Johann S. Bach
... do not mark sentence boundaries.  And sometimes sentences
... can start with non-capitalized words.  i is a good variable
... name.
... '''
>>> sent_detector = nltk.data.load('tokenizers/punkt/english.pickle')
>>> print('\n-----\n'.join(sent_detector.tokenize(text.strip())))
Punkt knows that the periods in Mr. Smith and Johann S. Bach
do not mark sentence boundaries.
-----
And sometimes sentences
can start with non-capitalized words.
-----
i is a good variable
name.
```



As used in the statute, "'act in furtherance of a person's right of petition or free speech under the United States or California Constitution in connection with a public issue' includes: (1) any written or oral statement or writing made before a legislative, executive, or judicial proceeding, or any other official proceeding authorized by law; (2) any written or oral statement or writing made in connection with an issue under consideration or review by a legislative, executive, or judicial body, or any other official proceeding authorized by law; (3) any written or oral statement or writing made in a place open to the public or a public forum in connection with an issue of public interest; (4) or any other conduct in furtherance of the exercise of the constitutional right of petition or the constitutional right of free speech in connection with a public issue or an issue of public interest." (§425.16, subd. (e), italics added; see Briggs v. Eden Council for Hope & Opportunity (1999) 19 Cal. 4th 1106, 1117-1118, 1123 [81 Cal.Rptr.2d 471, 969 P.2d 564] [discussing types of statements covered by anti-SLAPP statute].)

Figure 1: A sample passage from a legal decision. It contains a very long and complex sentence and a citation sentence. The first sentence contains a quotation which in turn contains another quotation and list. The second sentence illustrates the use of periods that are not sentence ending.

Sentence Detection

Problems with Pre-trained Machine Learning Models

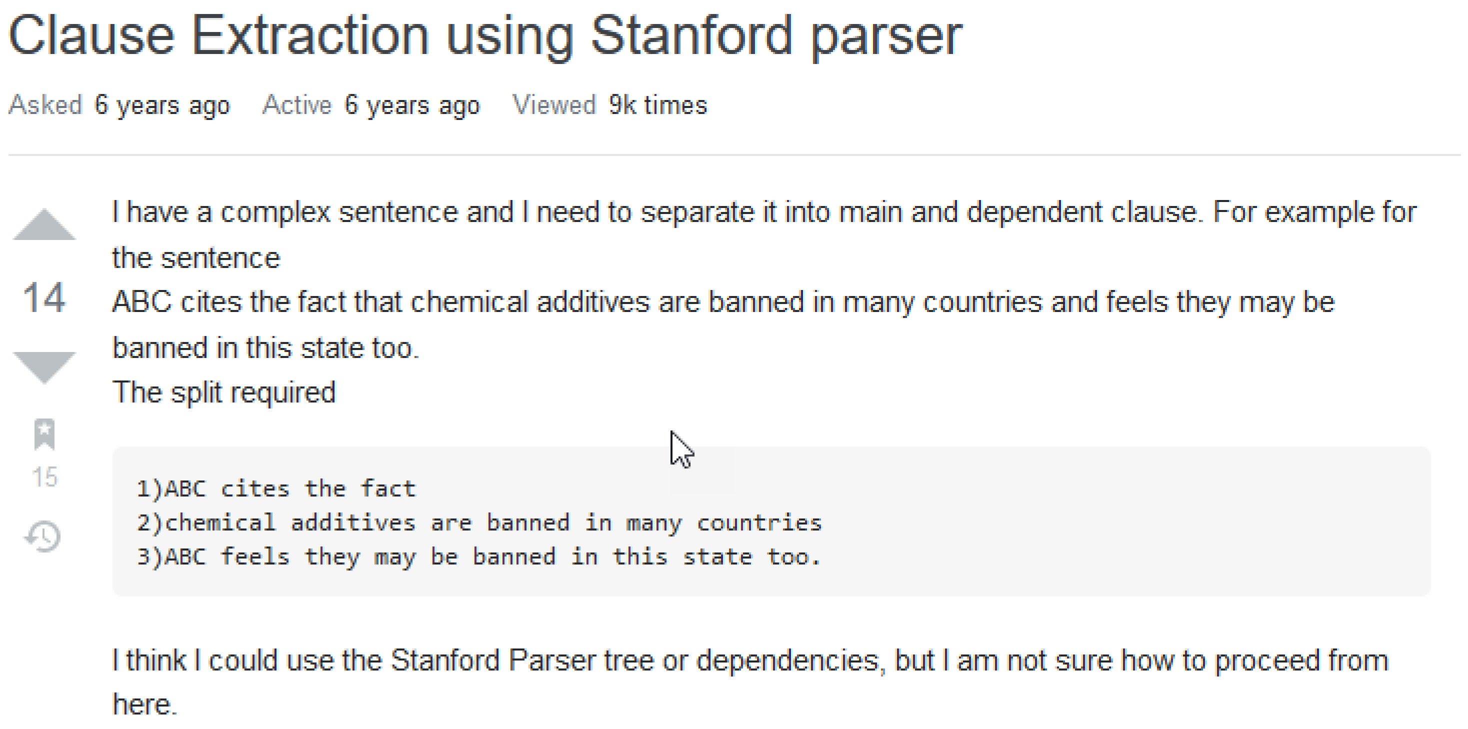
1. Sentence structure differs among languages (e.g., English vs Arabic, vs Chinese, etc.
2. Sentence structure can differ depending on the problem domains

Source: Sentence boundary detection in legal texts
http://web.stanford.edu/class/cs224n/reports/custom/report14.pdf

Clause Detection in a Sentence

Problem

1. A complex sentence can be multiple parts, each part denoting specific theme.
   - "It's great that I can use GSON for JSON parsing, but I like Jackson more for performance."
2. Each part is denoted as a clause
3. Clause detection is an open-ended question in natural language processing



Clause Extraction using Stanford parser

Asked 6 years ago Active 6 years ago Viewed 9k times

14

15

I have a complex sentence and I need to separate it into main and dependent clause. For example for the sentence
ABC cites the fact that chemical additives are banned in many countries and feels they may be banned in this state too.
The split required

```
1)ABC cites the fact
2)chemical additives are banned in many countries
3)ABC feels they may be banned in this state too.
```

I think I could use the Stanford Parser tree or dependencies, but I am not sure how to proceed from here.

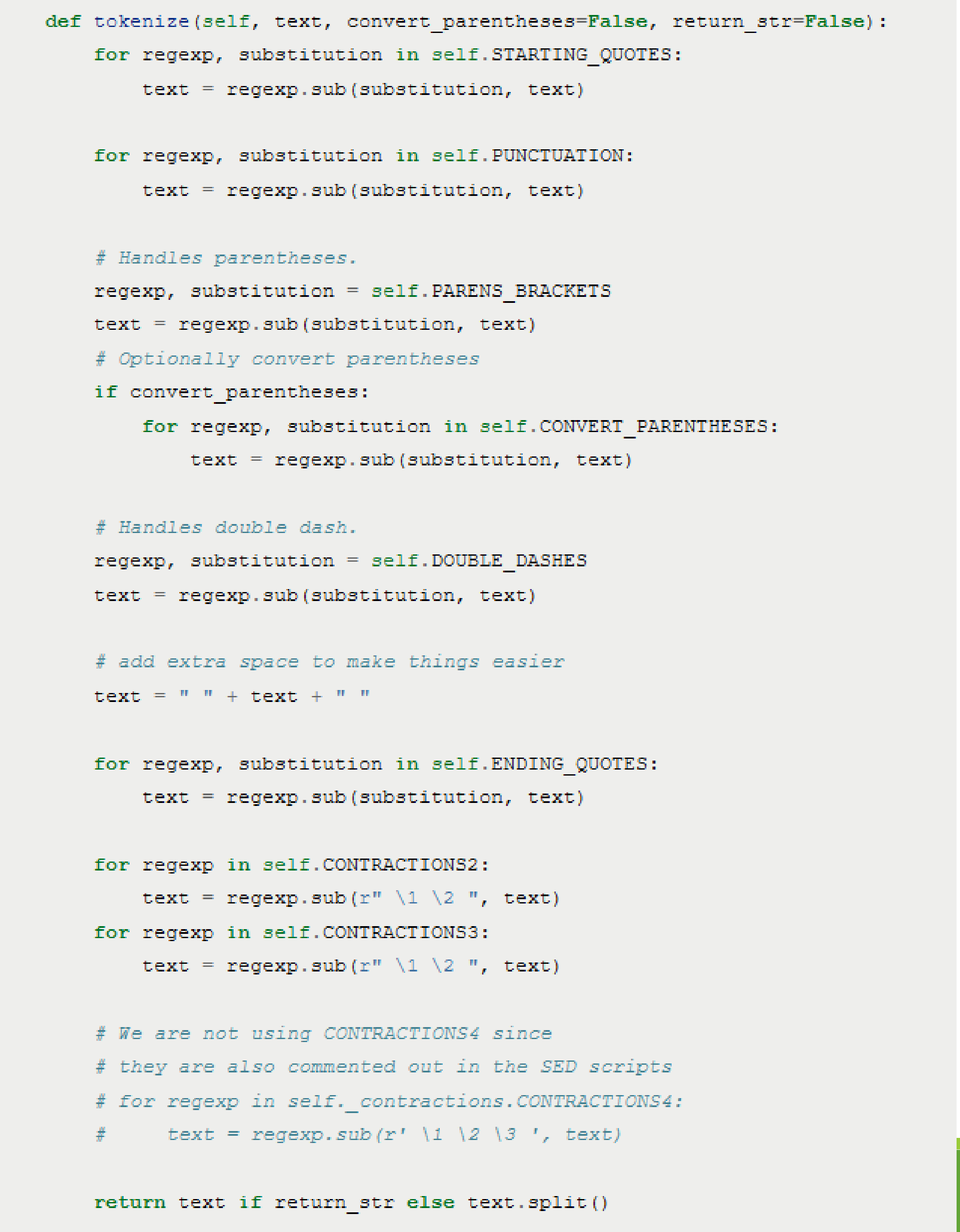
Source:https://stackoverflow.com/questions/26070245/clause-extraction-using-stanford-parser

Word Detection

Wikipedia: https://en.wikipedia.org/wiki/Text_segmentation#Word_segmentation

Word segmentation is the problem of dividing a string of written language into its component words. In English and many other languages using some form of the Latin alphabet, the space is a good approximation of a word divider (word delimiter), although this concept has limits because of the variability with which languages emically regard collocations and compounds. Many English compound nouns are variably written (for example, *ice box = ice-box = icebox*; *pig sty = pig-sty = pigsty*) with a corresponding variation in whether speakers think of them as noun phrases or single nouns; there are trends in how norms are set, such as that open compounds often tend eventually to solidify by widespread convention, but variation remains systemic. In contrast, German compound nouns show less orthographic variation, with solidification being a stronger norm.
However, the equivalent to the word space character is not found in all written scripts, and without it word segmentation is a difficult problem. Languages which do not have a trivial word segmentation process include Chinese, Japanese, where sentences but not words are delimited, Thai and Lao, where phrases and sentences but not words are delimited, and Vietnamese, where syllables but not words are delimited.

Word Detection



```
def tokenize(self, text, convert_parentheses=False, return_str=False):
    for regexp, substitution in self.STARTING_QUOTES:
        text = regexp.sub(substitution, text)

    for regexp, substitution in self.PUNCTUATION:
        text = regexp.sub(substitution, text)

    # Handles parentheses.
    regexp, substitution = self.PARENS_BRACKETS
    text = regexp.sub(substitution, text)
    # Optionally convert parentheses
    if convert_parentheses:
        for regexp, substitution in self.CONVERT_PARENTHESES:
            text = regexp.sub(substitution, text)

    # Handles double dash.
    regexp, substitution = self.DOUBLE_DASHES
    text = regexp.sub(substitution, text)

    # add extra space to make things easier
    text = " " + text + " "

    for regexp, substitution in self.ENDING_QUOTES:
        text = regexp.sub(substitution, text)

    for regexp in self.CONTRACTIONS2:
        text = regexp.sub(r" \1 \2 ", text)
    for regexp in self.CONTRACTIONS3:
        text = regexp.sub(r" \1 \2 ", text)

    # We are not using CONTRACTIONS4 since
    # they are also commented out in the SED scripts
    # for regexp in self._contractions.CONTRACTIONS4:
    #     text = regexp.sub(r' \1 \2 \3 ', text)

    return text if return_str else text.split()
```

Regular Expression Based

1. NLTK Punkt word Tokenizer: https://www.nltk.org/api/nltk.tokenize.html
2. https://www.nltk.org/_modules/nltk/tokenize/destructive.html#NLTKWordTokenizer
3. Other NLTK word detectors in **nltk.tokenize.regexp module**
    - RegexpTokenizer
    - BlanklineTokenizer
    - wordpunkt_tokenize
    - regexp_tokenize (allows setting your own regular expression pattern)
    - WhitespaceTokenizer
    - …

Noisy Text Detection

Wikipedia: https://en.wikipedia.org/wiki/Noisy_text_analytics

Noisy text analytics is a process of information extraction whose goal is to automatically extract structured or semi-structured information from noisy unstructured text data. While Text analytics is a growing and mature field that has great value because of the huge amounts of data being produced, processing of noisy text is gaining in importance because a lot of common applications produce noisy text data. Noisy unstructured text data is found in informal settings such as online chat, text messages, e-mails, message boards, newsgroups, blogs, wikis and web pages. Also, text produced by processing spontaneous speech using automatic speech recognition and printed or handwritten text using optical character recognition contains processing noise. Text produced under such circumstances is typically highly noisy containing spelling errors, abbreviations, non-standard words, false starts, repetitions, missing punctuations, missing letter case information, pause filling words such as “um” and “uh” and other texting and speech disfluencies. Such text can be seen in large amounts in contact centers, chat rooms, optical character recognition (OCR) of text documents, short message service (SMS) text, etc. Documents with historical language can also be considered noisy with respect to today's knowledge about the language. Such text contains important historical, religious, ancient medical knowledge that is useful. The nature of the noisy text produced in all these contexts warrants moving beyond traditional text analysis techniques.

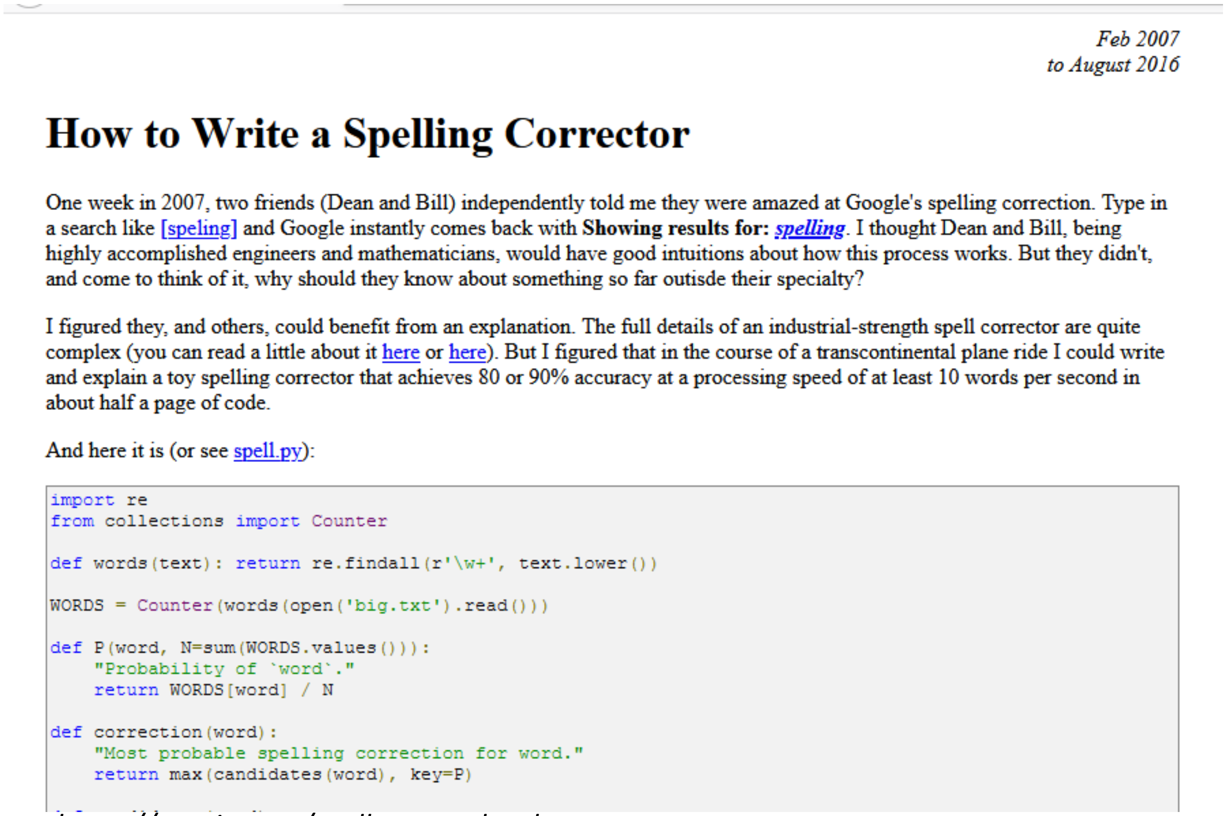
Noisy Text Detection

Detect Typo/Misspelling

It is just hapenning now!

Almost every editor now comes with built in spellchecker. Yet, typos are prevalent.

Spell correction in Google search engine – a very basic overview by Peter Norvig, Director of Research (Google)



Feb 2007
to August 2016

How to Write a Spelling Corrector

One week in 2007, two friends (Dean and Bill) independently told me they were amazed at Google's spelling correction. Type in a search like [speling] and Google instantly comes back with **Showing results for:** ***spelling***. I thought Dean and Bill, being highly accomplished engineers and mathematicians, would have good intuitions about how this process works. But they didn't, and come to think of it, why should they know about something so far outisde their specialty?

I figured they, and others, could benefit from an explanation. The full details of an industrial-strength spell corrector are quite complex (you can read a little about it here or here). But I figured that in the course of a transcontinental plane ride I could write and explain a toy spelling corrector that achieves 80 or 90% accuracy at a processing speed of at least 10 words per second in about half a page of code.

And here it is (or see spell.py):

```
import re
from collections import Counter

def words(text): return re.findall(r'\w+', text.lower())

WORDS = Counter(words(open('big.txt').read()))

def P(word, N=sum(WORDS.values())):
    "Probability of `word`."
    return WORDS[word] / N

def correction(word):
    "Most probable spelling correction for word."
    return max(candidates(word), key=P)
```

https://norvig.com/spell-correct.html

For each word:

1. check it against a dictionary of correct words.
2. is the word found in the dictionary?
3. no, then it could be a typo
4. use Levenshtein distance algorithm to find the closest candidate to replace it

Noisy Text Detection

Detect Typo/Misspelling using Levenshtein Distance

Wikipedia:
In information theory, linguistics and computer science, the **Levenshtein distance** is a string metric for measuring the difference between two sequences. Informally, the Levenshtein distance between two words is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other. Levenshtein distance may also be referred to as *edit distance.* It is closely related to pairwise string alignments.
Mathematically the Levenshtein distance between two strings a and b (of length $|a|$ and $|b|$) is given by $lev_{a,b}(|a|,|b|)$ where:

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j)=0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j)+1 \\ \operatorname{lev}_{a,b}(i,j-1)+1 \\ \operatorname{lev}_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Here, $1(a_i \neq b_i)$ is the indicator function equal to 0 when $a_i \neq b_i$and 1 otherwise.
$lev_{a,b}(|a|,|b|)$ is the distance between the first i characters of a and the first j characters of b

Noisy Text Detection

Detect Typo/Misspelling using Levenshtein Distance

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Here, $1(a_i \neq b_i)$ is the indicator function equal to 0 when $a_i \neq b_i$and 1 otherwise.
$lev_{a,b}(|a|,|b|)$ is the distance between the first i characters of a and the first j characters of b

3 Edit Operations:

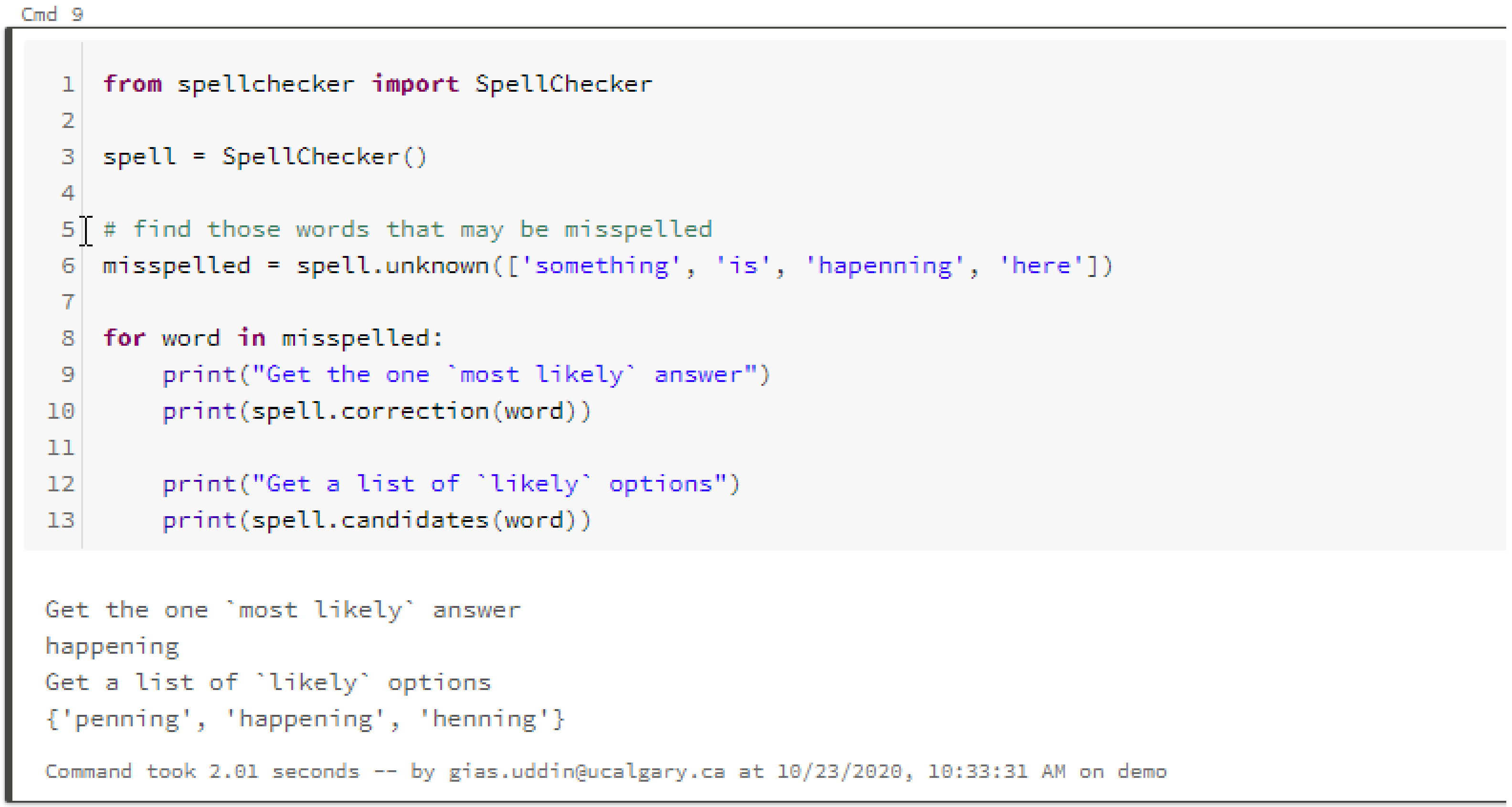
1. Insertions: car > car**t**
2. Deletions: ca**r**t > cat
3. Substitutions: **c**art > **d**art

A good example of edit distance in YouTube:
https://www.youtube.com/watch?v=M3MQt9GJPW0

Noisy Text Detection

Detect Typo/Misspelling using Levenshtein Distance in Python

pip install pyspellchecker


Cmd 9

```python
from spellchecker import SpellChecker

spell = SpellChecker()

# find those words that may be misspelled
misspelled = spell.unknown(['something', 'is', 'hapenning', 'here'])

for word in misspelled:
    print("Get the one `most likely` answer")
    print(spell.correction(word))

    print("Get a list of `likely` options")
    print(spell.candidates(word))
```

```
Get the one `most likely` answer
happening
Get a list of `likely` options
{'penning', 'happening', 'henning'}
```

Command took 2.01 seconds -- by gias.uddin@ucalgary.ca at 10/23/2020, 10:33:31 AM on demo

Noisy Text Detection

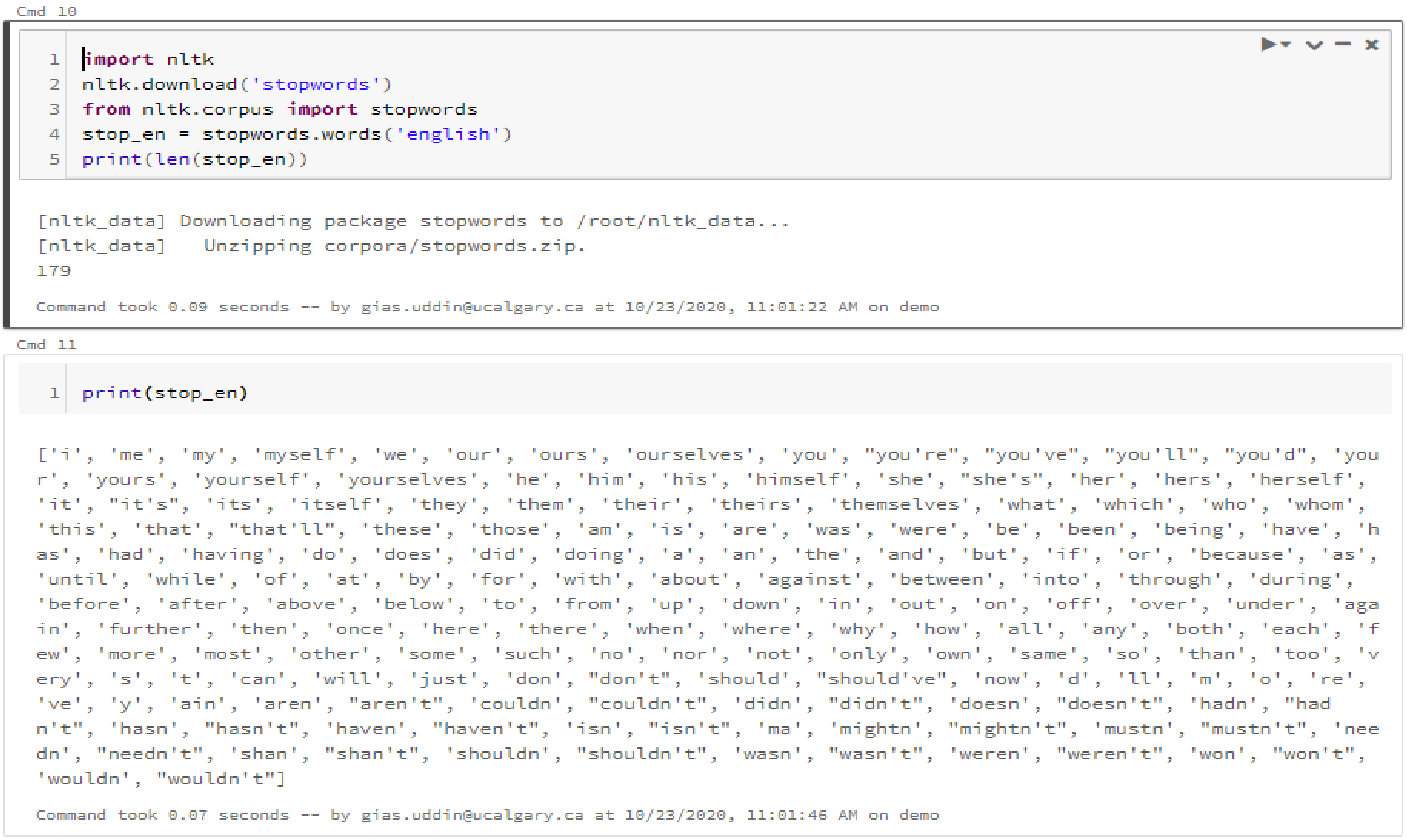
Stop words

Wikipedia:
In computing, **stop words** are words which are filtered out before or after processing of natural language data (text).[1] Though "stop words" usually refers to the most common words in a language, there is no single universal list of stop words used by all natural language processing tools, and indeed not all tools even use such a list. Some tools specifically avoid removing these stop words to support phrase search.
Any group of words can be chosen as the stop words for a given purpose. For some search engines, these are some of the most common, short function words, such as *the*, *is*, *at*, *which*, and *on*. In this case, stop words can cause problems when searching for phrases that include them, particularly in names such as "The Who", "The The", or "Take That". Other search engines remove some of the most common words—including lexical words, such as "want"—from a query in order to improve performance.[2]

1. NLTK's list of English stop words: https://gist.github.com/sebleier/554280
2. Common search and database (e.g., MySQL) stop words: https://www.ranks.nl/stopwords
3. Language specific stopwords: https://cran.r-project.org/web/packages/stopwords/stopwords.pdf

Noisy Text Detection

Stop words removal using python



Cmd 10

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stop_en = stopwords.words('english')
print(len(stop_en))
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
179
```

Command took 0.09 seconds -- by gias.uddin@ucalgary.ca at 10/23/2020, 11:01:22 AM on demo

Cmd 11

```
print(stop_en)
```

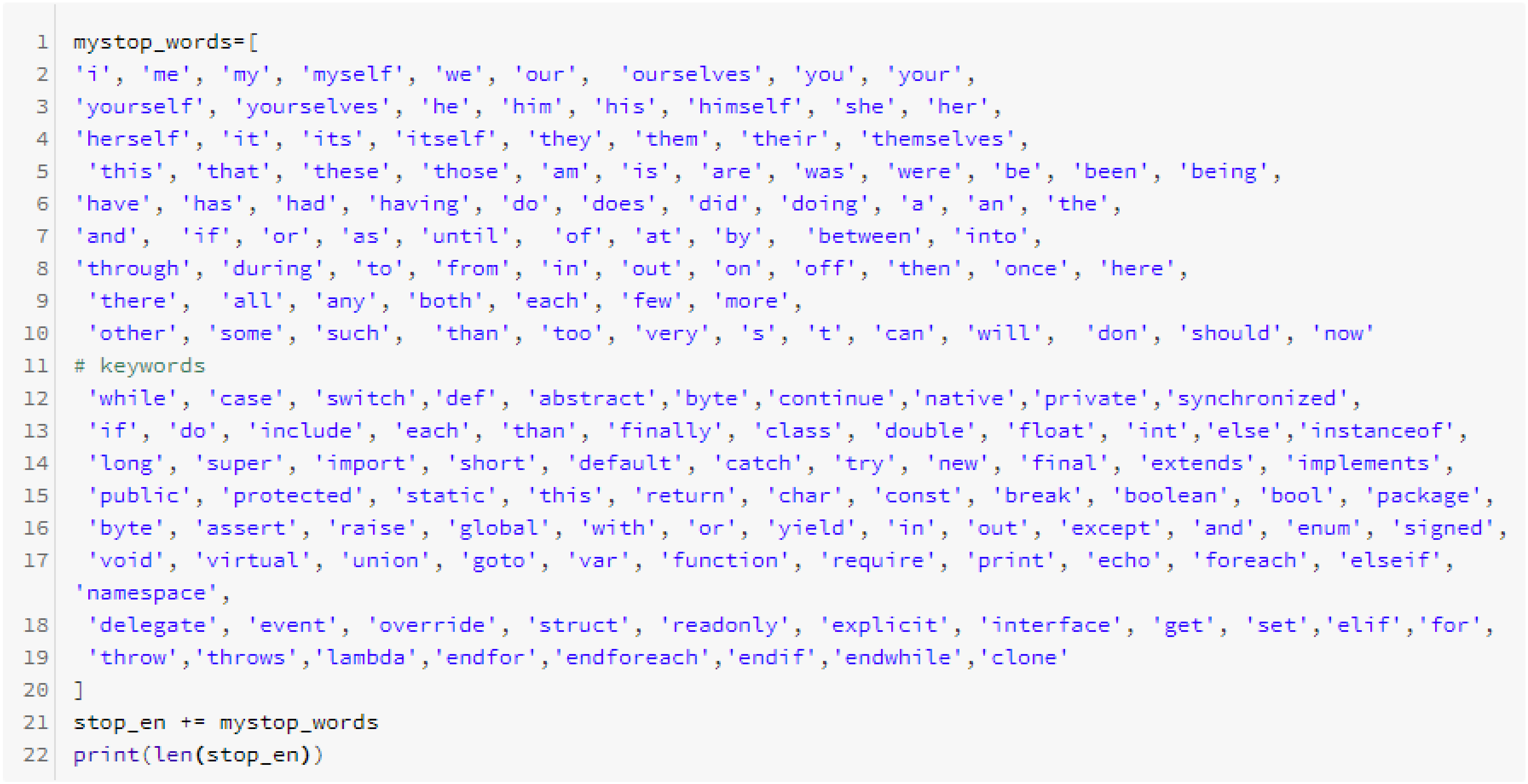
```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'you
r', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself',
'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom',
'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'h
as', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', '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', 'aga
in', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'f
ew', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'v
ery', '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', "had
n't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'nee
dn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't",
'wouldn', "wouldn't"]
```

Command took 0.07 seconds -- by gias.uddin@ucalgary.ca at 10/23/2020, 11:01:46 AM on demo

Noisy Text Detection

Stop words removal using python

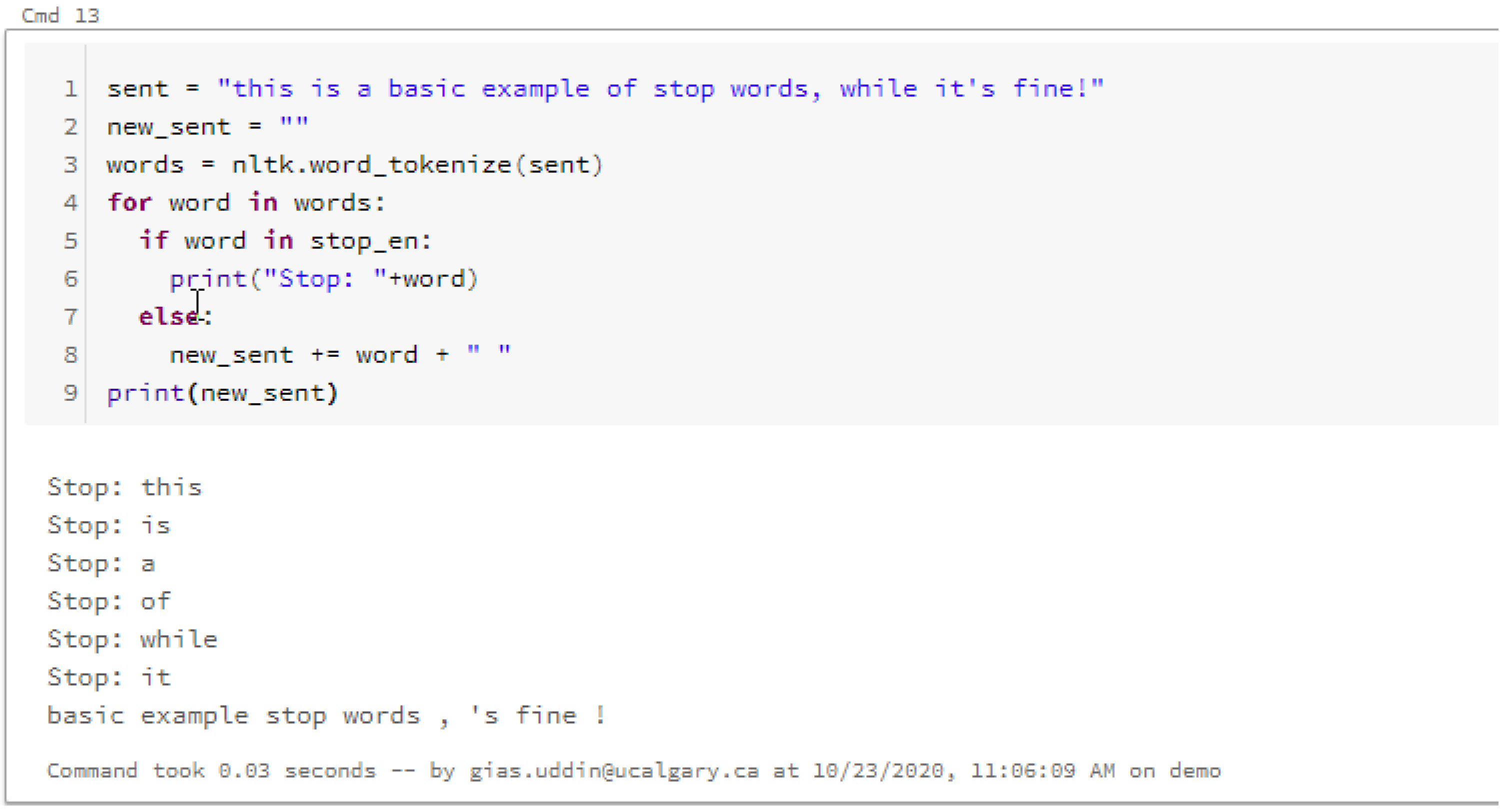
We can add our personalized list of stop words into the nltk stop words during our analysis



```python
mystop_words=[
'i', 'me', 'my', 'myself', 'we', 'our',  'ourselves', 'you', 'your',
'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', 'her',
'herself', 'it', 'its', 'itself', 'they', 'them', 'their', 'themselves',
 'this', 'that', 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being',
'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the',
'and',  'if', 'or', 'as', 'until',  'of', 'at', 'by',  'between', 'into',
'through', 'during', 'to', 'from', 'in', 'out', 'on', 'off', 'then', 'once', 'here',
 'there',  'all', 'any', 'both', 'each', 'few', 'more',
 'other', 'some', 'such',  'than', 'too', 'very', 's', 't', 'can', 'will',  'don', 'should', 'now'
# keywords
 'while', 'case', 'switch','def', 'abstract','byte','continue','native','private','synchronized',
 'if', 'do', 'include', 'each', 'than', 'finally', 'class', 'double', 'float', 'int','else','instanceof',
 'long', 'super', 'import', 'short', 'default', 'catch', 'try', 'new', 'final', 'extends', 'implements',
 'public', 'protected', 'static', 'this', 'return', 'char', 'const', 'break', 'boolean', 'bool', 'package',
 'byte', 'assert', 'raise', 'global', 'with', 'or', 'yield', 'in', 'out', 'except', 'and', 'enum', 'signed',
 'void', 'virtual', 'union', 'goto', 'var', 'function', 'require', 'print', 'echo', 'foreach', 'elseif',
'namespace',
 'delegate', 'event', 'override', 'struct', 'readonly', 'explicit', 'interface', 'get', 'set','elif','for',
 'throw','throws','lambda','endfor','endforeach','endif','endwhile','clone'
]
stop_en += mystop_words
print(len(stop_en))
```

Noisy Text Detection

Stop words removal using python

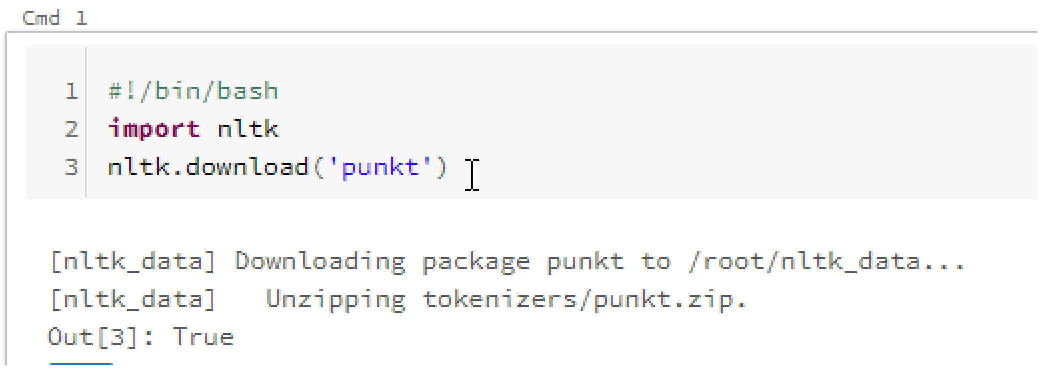

Cmd 13

```
sent = "this is a basic example of stop words, while it's fine!"
new_sent = ""
words = nltk.word_tokenize(sent)
for word in words:
  if word in stop_en:
    print("Stop: "+word)
  else:
    new_sent += word + " "
print(new_sent)
```

```
Stop: this
Stop: is
Stop: a
Stop: of
Stop: while
Stop: it
basic example stop words , 's fine !
```

Command took 0.03 seconds -- by gias.uddin@ucalgary.ca at 10/23/2020, 11:06:09 AM on demo

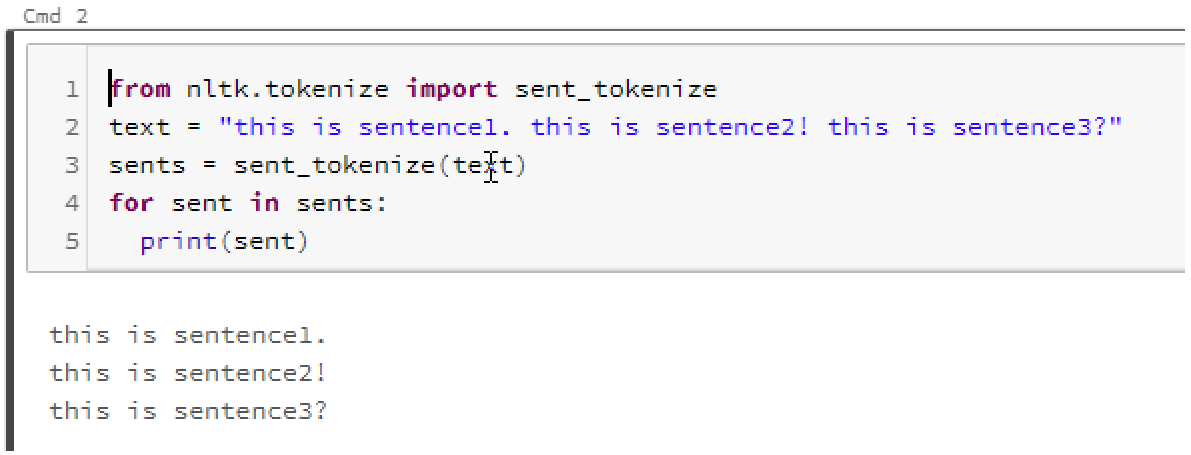
Use Case Based on Learning So Far - Word count



Cmd 1

```
#!/bin/bash
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
Out[3]: True
```

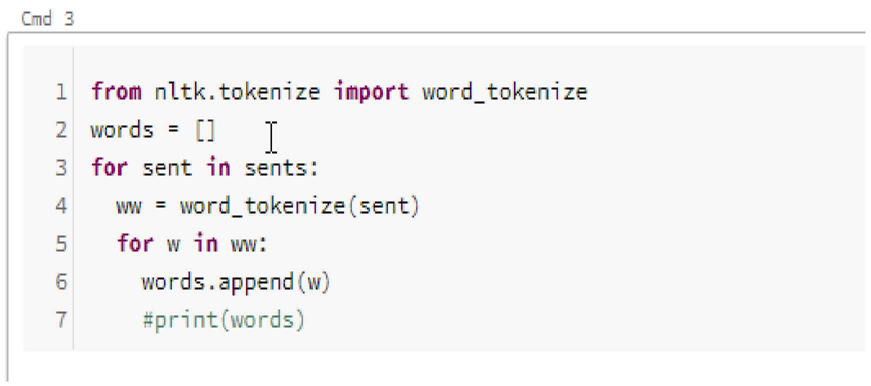


Cmd 2

```
from nltk.tokenize import sent_tokenize
text = "this is sentence1. this is sentence2! this is sentence3?"
sents = sent_tokenize(text)
for sent in sents:
  print(sent)
```

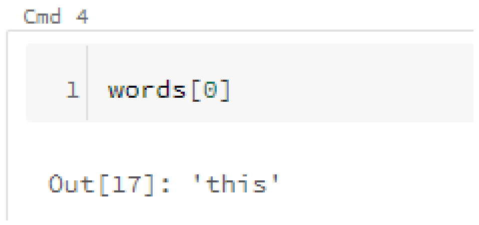
```
this is sentence1.
this is sentence2!
this is sentence3?
```

Use Case Based on Learning So Far - Word count



Cmd 3

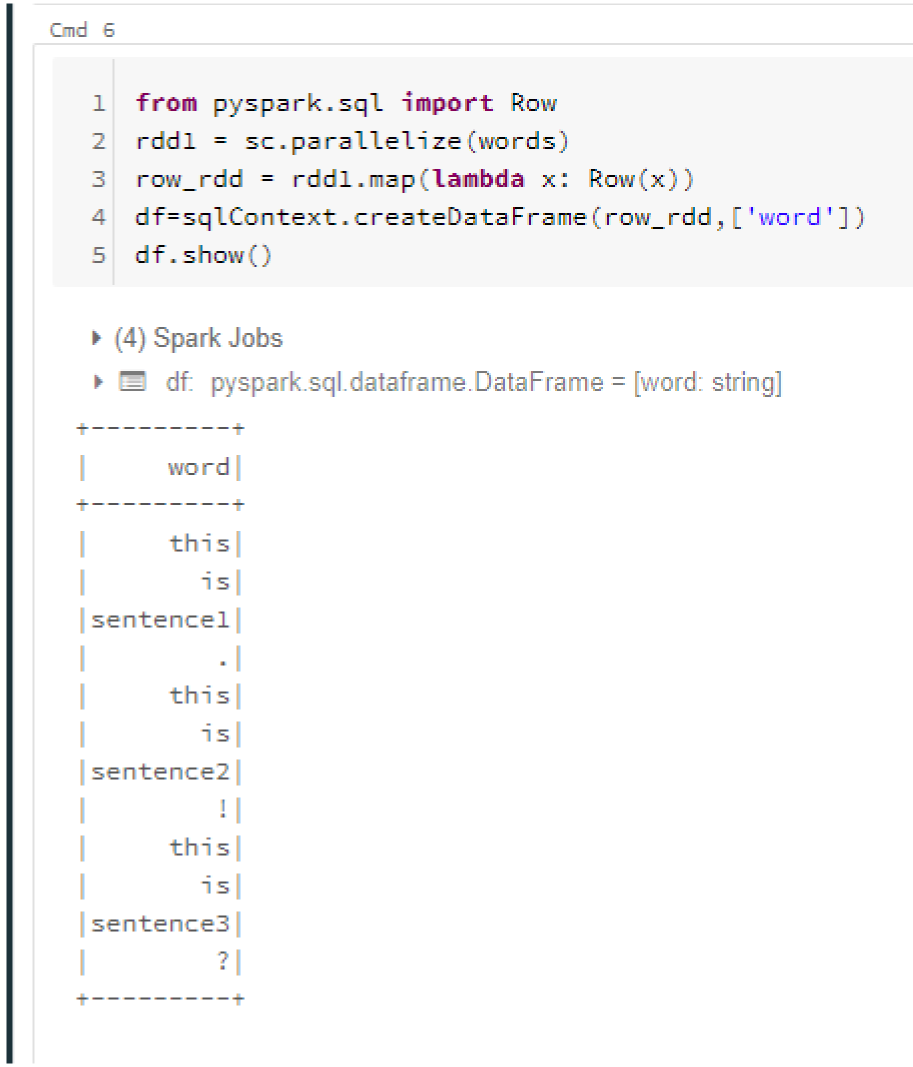
```
from nltk.tokenize import word_tokenize
words = []
for sent in sents:
  ww = word_tokenize(sent)
  for w in ww:
    words.append(w)
    #print(words)
```



Cmd 4

```
words[0]
```

```
Out[17]: 'this'
```



Cmd 6

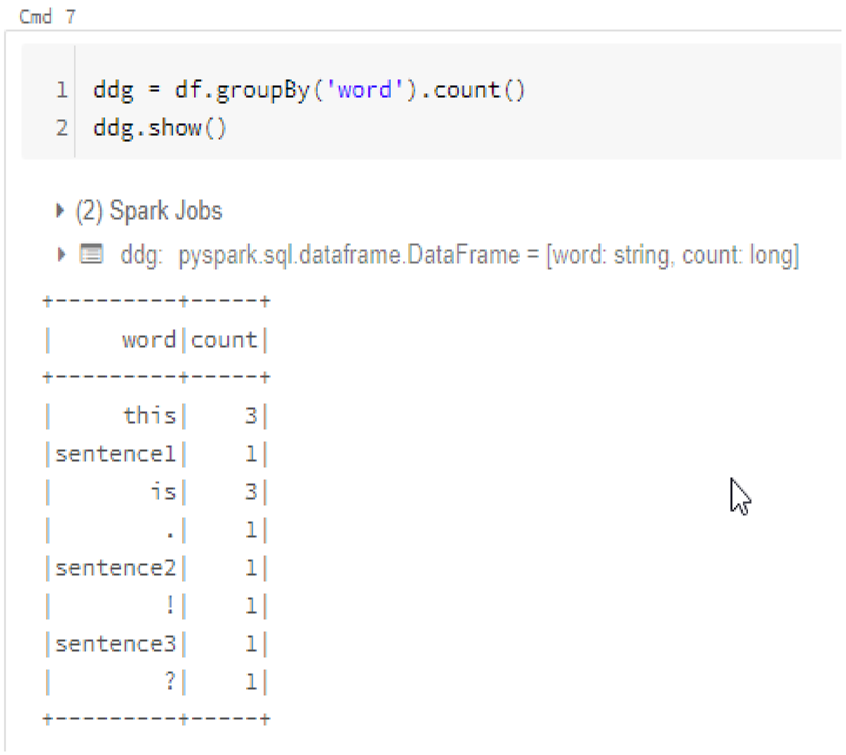
```
from pyspark.sql import Row
rdd1 = sc.parallelize(words)
row_rdd = rdd1.map(lambda x: Row(x))
df=sqlContext.createDataFrame(row_rdd,['word'])
df.show()
```

▸ (4) Spark Jobs

▸ df: pyspark.sql.dataframe.DataFrame = [word: string]

```
+---------+
|     word|
+---------+
|     this|
|       is|
|sentence1|
|        .|
|     this|
|       is|
|sentence2|
|        !|
|     this|
|       is|
|sentence3|
|        ?|
+---------+
```

Use Case Based on Learning So Far - Word count



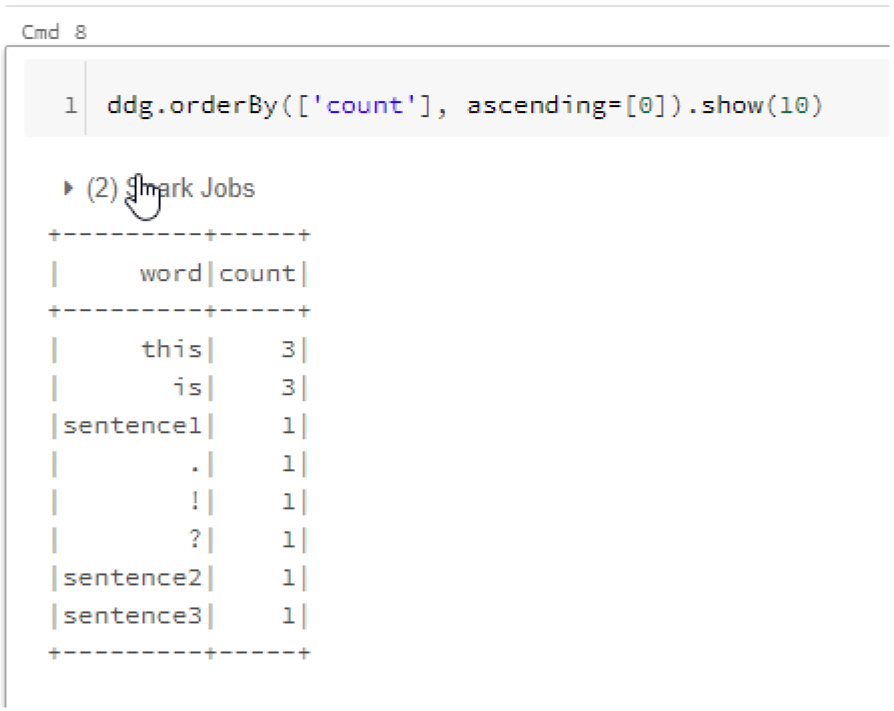
Cmd 7

```python
ddg = df.groupBy('word').count()
ddg.show()
```

▸ (2) Spark Jobs

▸ ddg: pyspark.sql.dataframe.DataFrame = [word: string, count: long]

```
+---------+-----+
|     word|count|
+---------+-----+
|     this|    3|
|sentence1|    1|
|       is|    3|
|        .|    1|
|sentence2|    1|
|        !|    1|
|sentence3|    1|
|        ?|    1|
+---------+-----+
```



Cmd 8

```python
ddg.orderBy(['count'], ascending=[0]).show(10)
```

▸ (2) Spark Jobs

```
+---------+-----+
|     word|count|
+---------+-----+
|     this|    3|
|       is|    3|
|sentence1|    1|
|        .|    1|
|        !|    1|
|        ?|    1|
|sentence2|    1|
|sentence3|    1|
+---------+-----+
```