Text Analysis

Tools and Techniques for Data Science

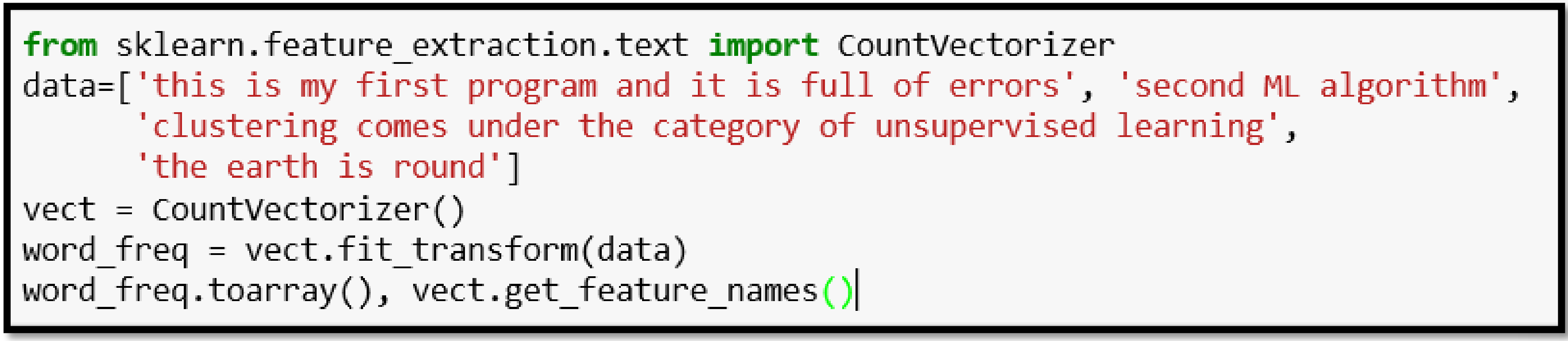
Murk Marvi

Outline

- Bag-of-words
- TF-IDF-continue
- Stemming
- Lemmatization

Text Analysis

- **Bag of words:** It is commonly used in methods of document classification where the frequency of (occurrence of) each word is used as a feature for training a classifier.



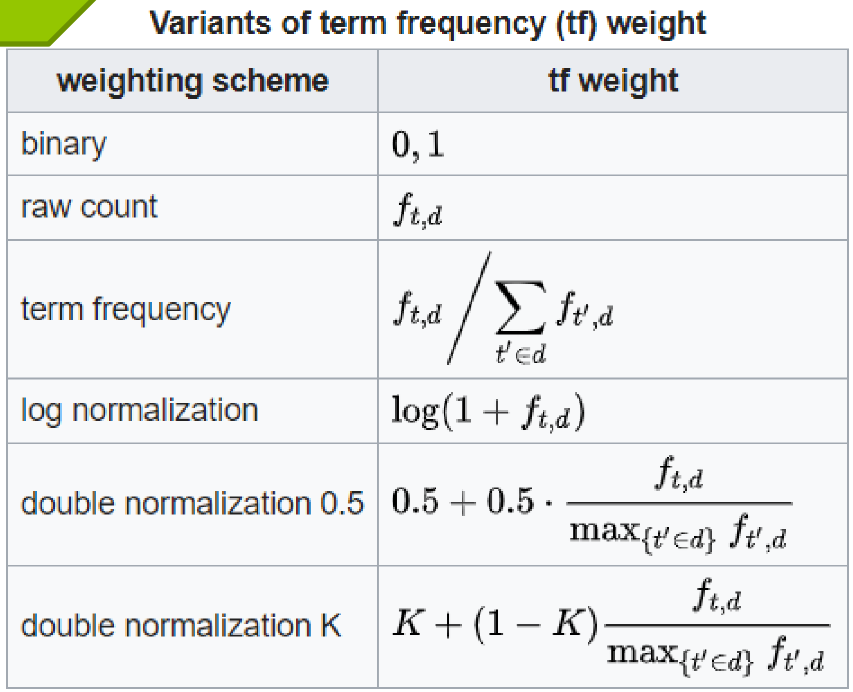
```
from sklearn.feature_extraction.text import CountVectorizer
data=['this is my first program and it is full of errors', 'second ML algorithm',
      'clustering comes under the category of unsupervised learning',
      'the earth is round']
vect = CountVectorizer()
word_freq = vect.fit_transform(data)
word_freq.toarray(), vect.get_feature_names()
```

Text Analysis

- **Term Frequency–Inverse Document Frequency (TF-IDF):** It is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.

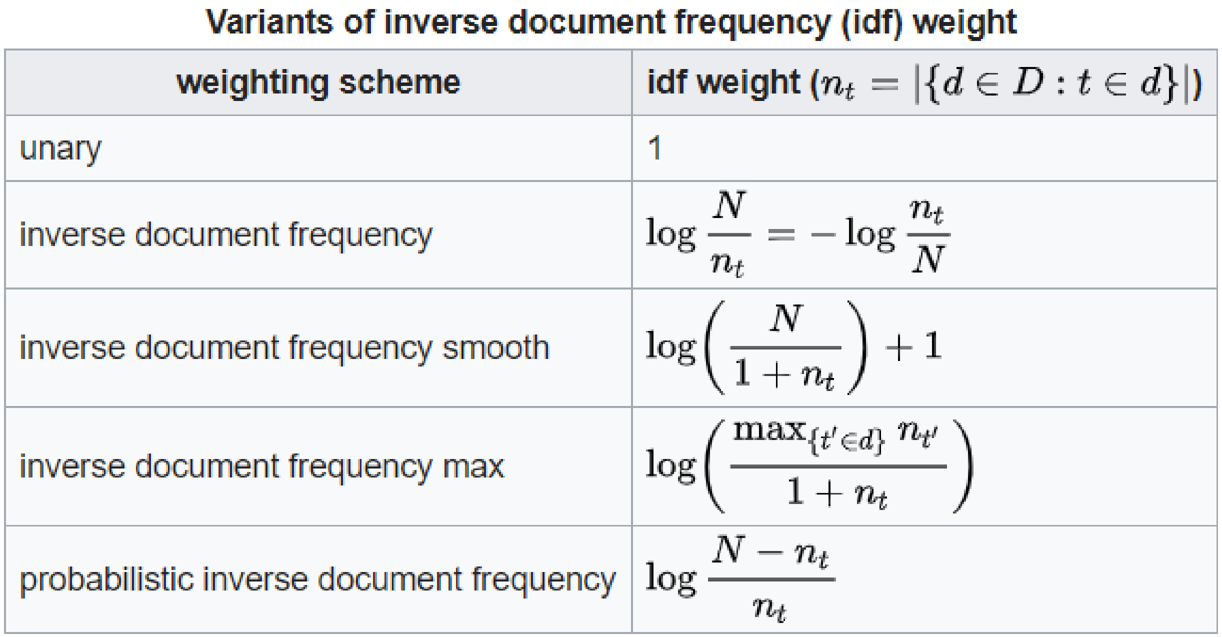
$$\boxed{\mathrm{tfidf}(t, d, D) = \mathrm{tf}(t, d) \cdot \mathrm{idf}(t, D)}$$

Text Analysis



Variants of term frequency (tf) weight

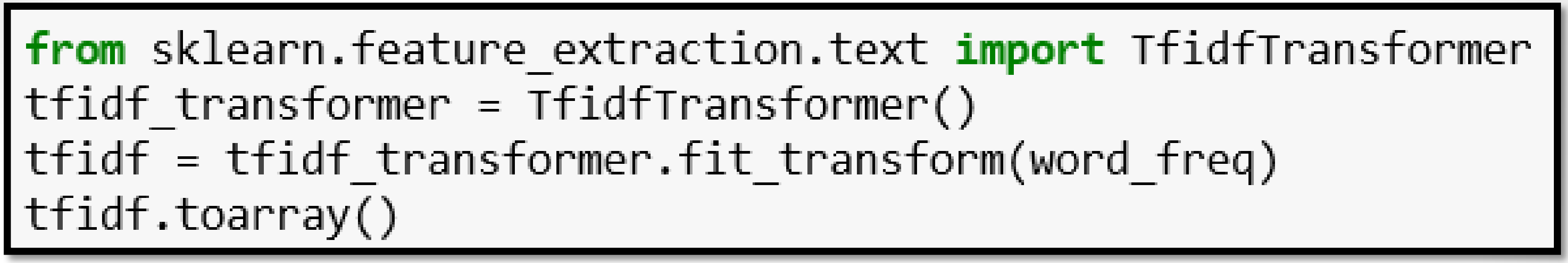
weighting scheme	tf weight
binary	$0, 1$
raw count	$f_{t,d}$
term frequency	$f_{t,d} \Bigg/ \sum_{t' \in d} f_{t',d}$
log normalization	$\log(1 + f_{t,d})$
double normalization 0.5	$0.5 + 0.5 \cdot \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$
double normalization K	$K + (1 - K)\frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$



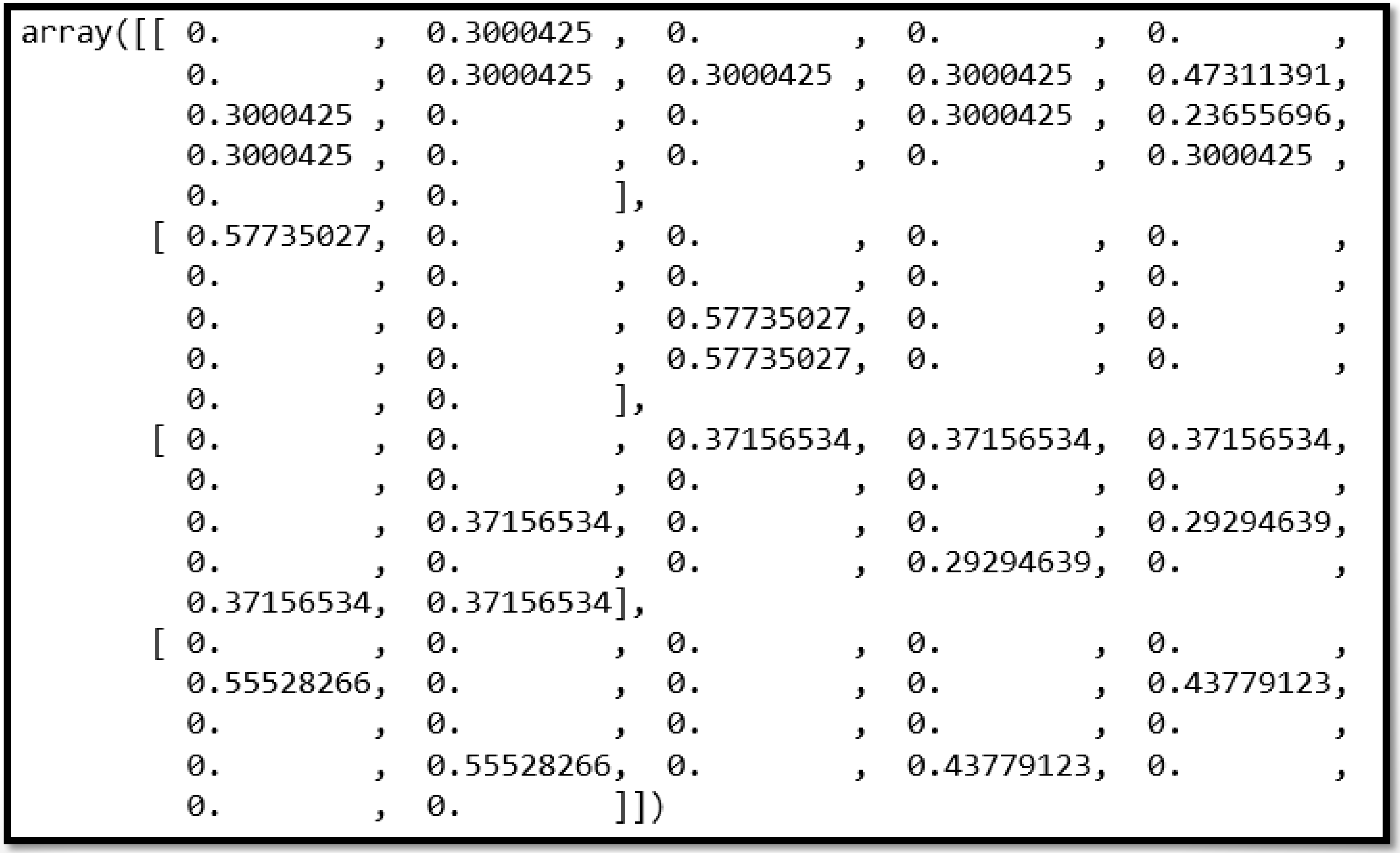
Variants of inverse document frequency (idf) weight

weighting scheme	idf weight ($n_t = \|\{d \in D : t \in d\}\|$)
unary	1
inverse document frequency	$\log \frac{N}{n_t} = -\log \frac{n_t}{N}$
inverse document frequency smooth	$\log \left(\frac{N}{1 + n_t} \right) + 1$
inverse document frequency max	$\log \left(\frac{\max_{\{t' \in d\}} n_{t'}}{1 + n_t} \right)$
probabilistic inverse document frequency	$\log \frac{N - n_t}{n_t}$

Text Analysis



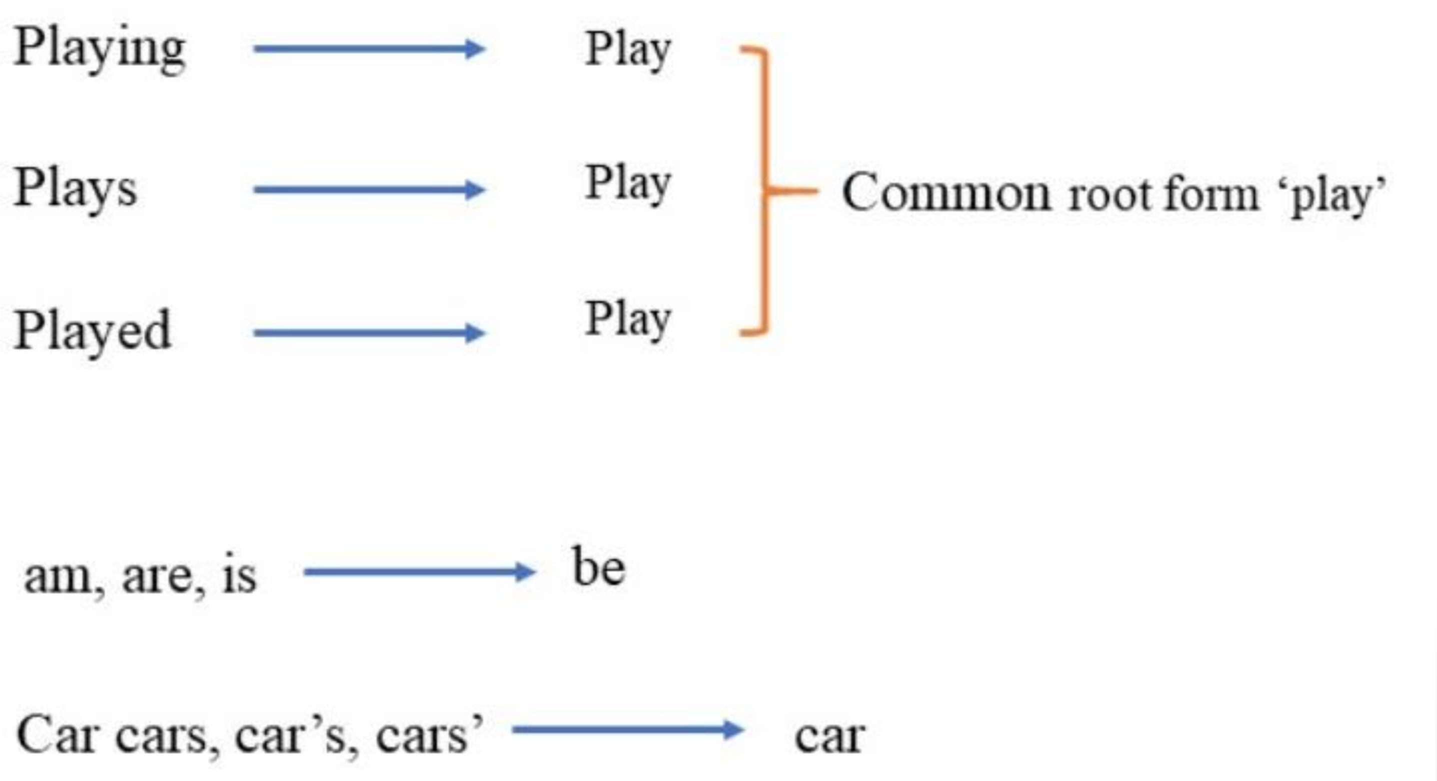
```
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer()
tfidf = tfidf_transformer.fit_transform(word_freq)
tfidf.toarray()
```



```
array([[ 0.        ,  0.3000425 ,  0.        ,  0.        ,  0.        ,
         0.        ,  0.3000425 ,  0.3000425 ,  0.3000425 ,  0.47311391,
         0.3000425 ,  0.        ,  0.        ,  0.3000425 ,  0.23655696,
         0.3000425 ,  0.        ,  0.        ,  0.        ,  0.3000425 ,
         0.        ,  0.        ],
       [ 0.57735027,  0.        ,  0.        ,  0.        ,  0.        ,
         0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
         0.        ,  0.        ,  0.57735027,  0.        ,  0.        ,
         0.        ,  0.        ,  0.57735027,  0.        ,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.37156534,  0.37156534,  0.37156534,
         0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
         0.        ,  0.37156534,  0.        ,  0.        ,  0.29294639,
         0.        ,  0.        ,  0.        ,  0.29294639,  0.        ,
         0.37156534,  0.37156534],
       [ 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
         0.55528266,  0.        ,  0.        ,  0.        ,  0.43779123,
         0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
         0.        ,  0.55528266,  0.        ,  0.43779123,  0.        ,
         0.        ,  0.        ]])
```

Text Analysis

- **Inflected Language:** When a language contains words that are derived from another word as their use in the speech changes is called **Inflected Language**.



Playing → Play

Plays → Play

Played → Play

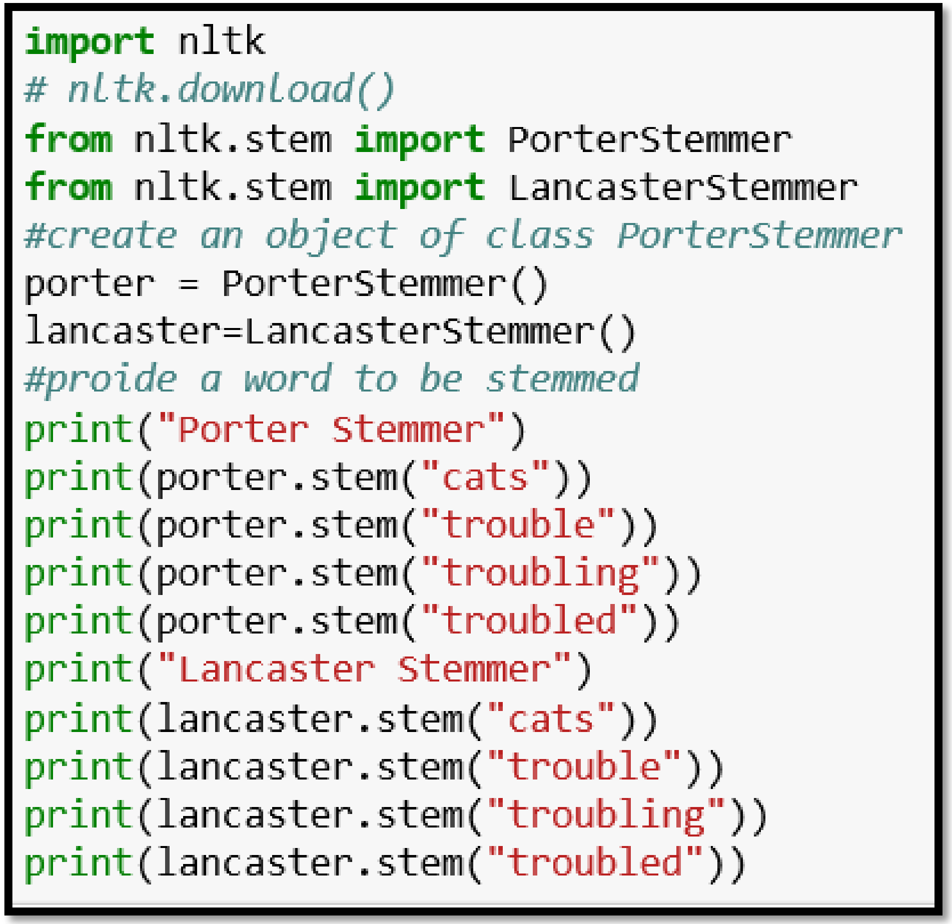
Common root form ‘play’

am, are, is → be

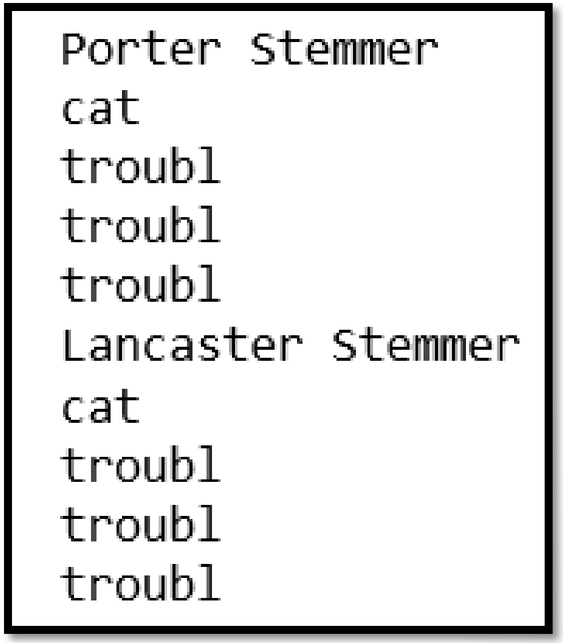
Car cars, car’s, cars’ → car

Text Analysis

- **Stemming:** It is the process of reducing inflection in words to their root forms such as mapping a group of words to the same stem even if the stem itself is not a valid word in the Language.

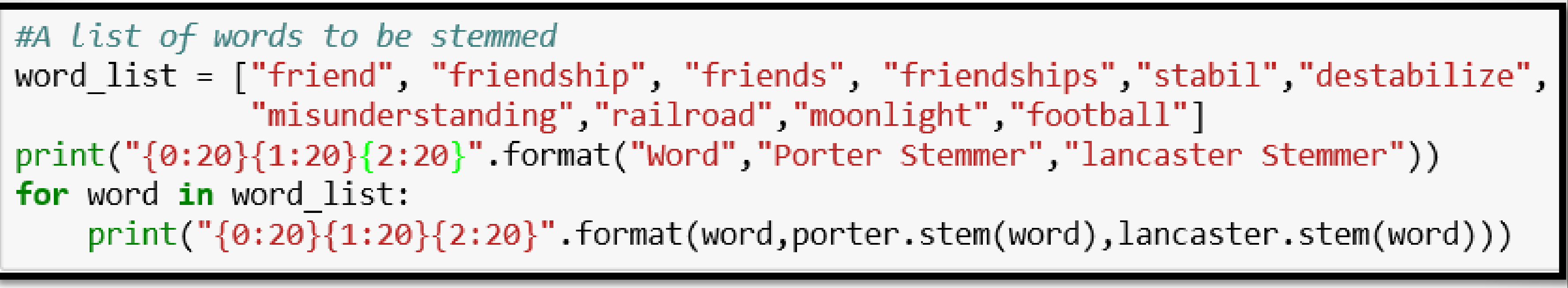


```python
import nltk
# nltk.download()
from nltk.stem import PorterStemmer
from nltk.stem import LancasterStemmer
#create an object of class PorterStemmer
porter = PorterStemmer()
lancaster=LancasterStemmer()
#proide a word to be stemmed
print("Porter Stemmer")
print(porter.stem("cats"))
print(porter.stem("trouble"))
print(porter.stem("troubling"))
print(porter.stem("troubled"))
print("Lancaster Stemmer")
print(lancaster.stem("cats"))
print(lancaster.stem("trouble"))
print(lancaster.stem("troubling"))
print(lancaster.stem("troubled"))
```

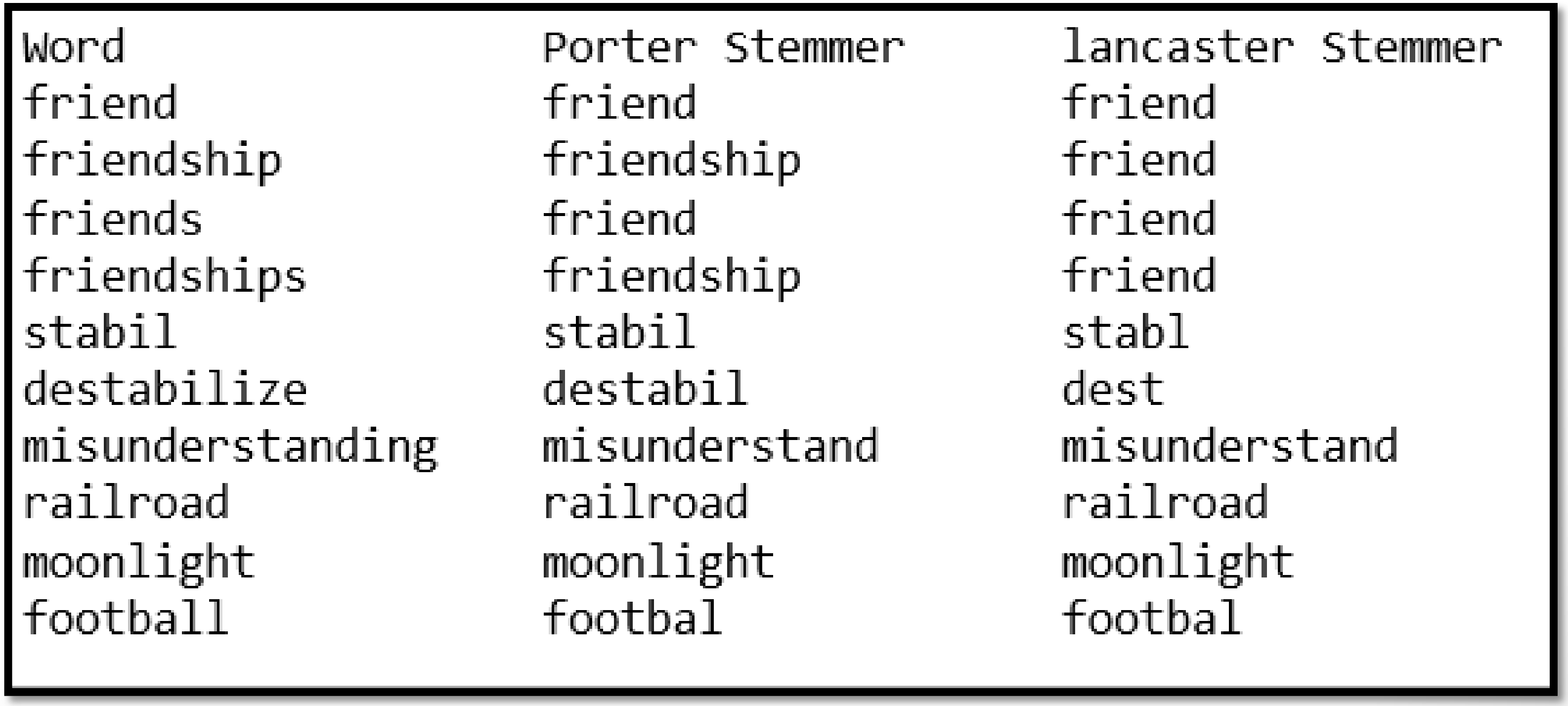


```
Porter Stemmer
cat
troubl
troubl
troubl
Lancaster Stemmer
cat
troubl
troubl
troubl
```

Text Analysis

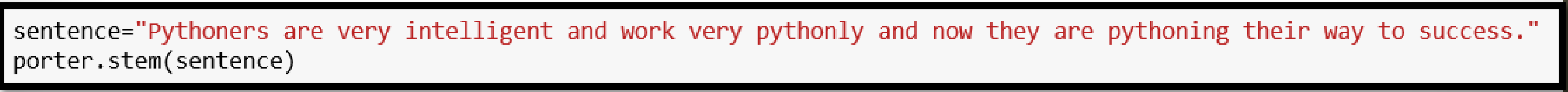


```python
#A list of words to be stemmed
word_list = ["friend", "friendship", "friends", "friendships","stabil","destabilize",
             "misunderstanding","railroad","moonlight","football"]
print("{0:20}{1:20}{2:20}".format("Word","Porter Stemmer","lancaster Stemmer"))
for word in word_list:
    print("{0:20}{1:20}{2:20}".format(word,porter.stem(word),lancaster.stem(word)))
```

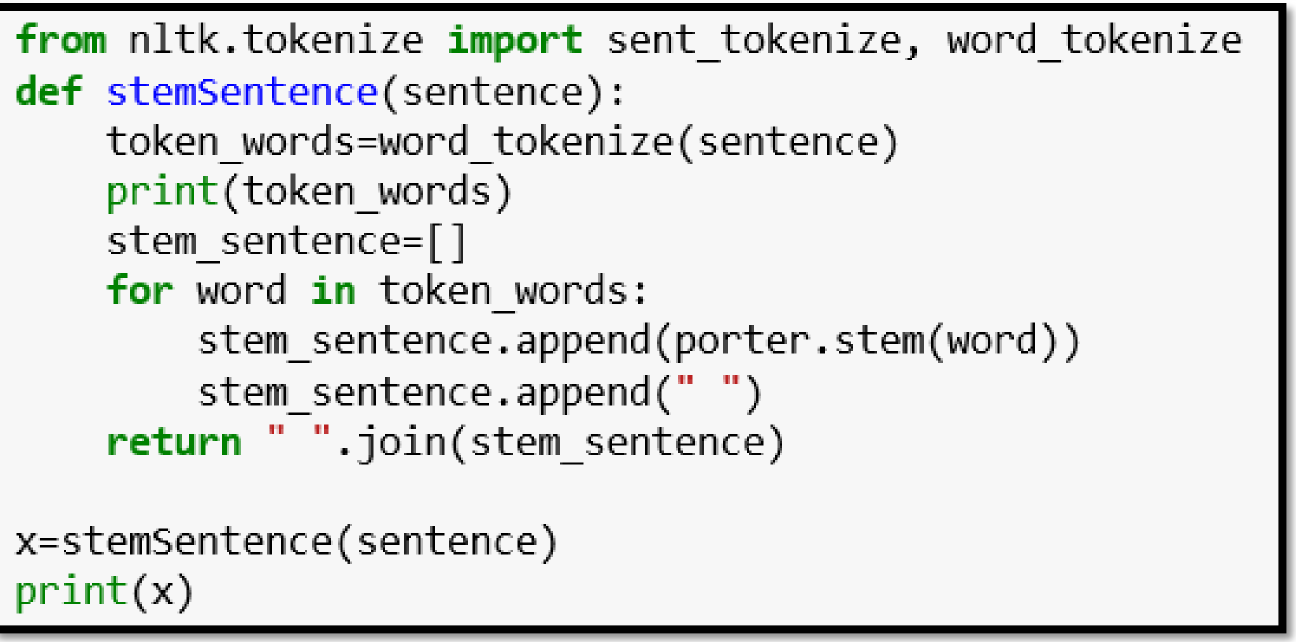


```
Word                Porter Stemmer      lancaster Stemmer
friend              friend              friend
friendship          friendship          friend
friends             friend              friend
friendships         friendship          friend
stabil              stabil              stabl
destabilize         destabil            dest
misunderstanding    misunderstand       misunderstand
railroad            railroad            railroad
moonlight           moonlight           moonlight
football            footbal             footbal
```

Text Analysis



```
sentence="Pythoners are very intelligent and work very pythonly and now they are pythoning their way to success."
porter.stem(sentence)
```

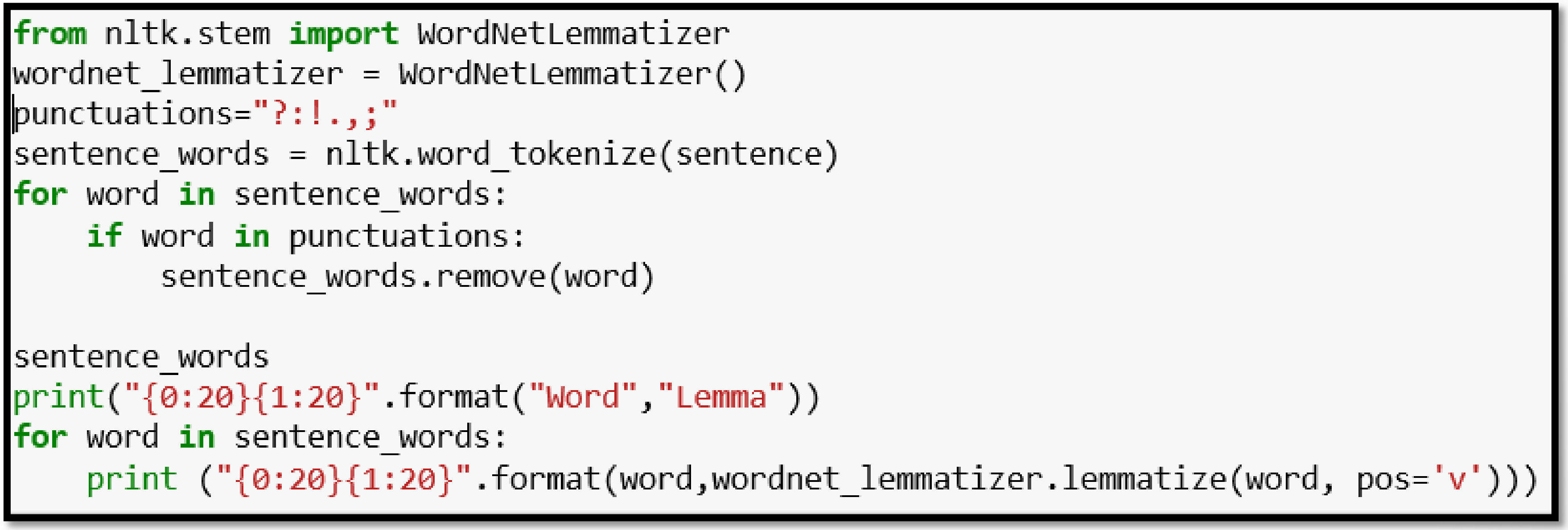


```
from nltk.tokenize import sent_tokenize, word_tokenize
def stemSentence(sentence):
    token_words=word_tokenize(sentence)
    print(token_words)
    stem_sentence=[]
    for word in token_words:
        stem_sentence.append(porter.stem(word))
        stem_sentence.append(" ")
    return " ".join(stem_sentence)

x=stemSentence(sentence)
print(x)
```

Text Analysis

- **Lemmatization:** The process of reducing different forms of a word to one single form, for example, reducing "builds", "building", or "built" to the lemma "build“.



```
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
punctuations="?:!.,;"
sentence_words = nltk.word_tokenize(sentence)
for word in sentence_words:
    if word in punctuations:
        sentence_words.remove(word)

sentence_words
print("{0:20}{1:20}".format("Word","Lemma"))
for word in sentence_words:
    print ("{0:20}{1:20}".format(word,wordnet_lemmatizer.lemmatize(word, pos='v')))
```