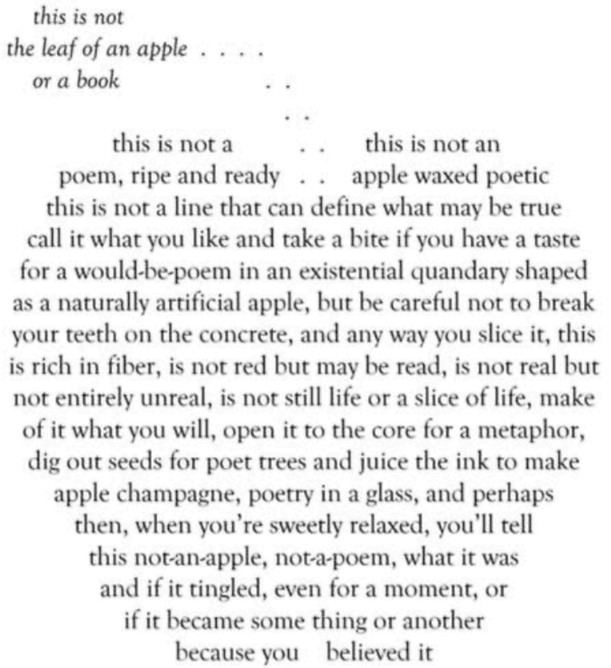
Intro to text mining - Text processing - 2

One should look for what is and not what he thinks should be. (Albert Einstein)

Chat question

- **How many times** does the word **“apple”** appear in the poem?
- **How long** did it take you to arrive at your answer?

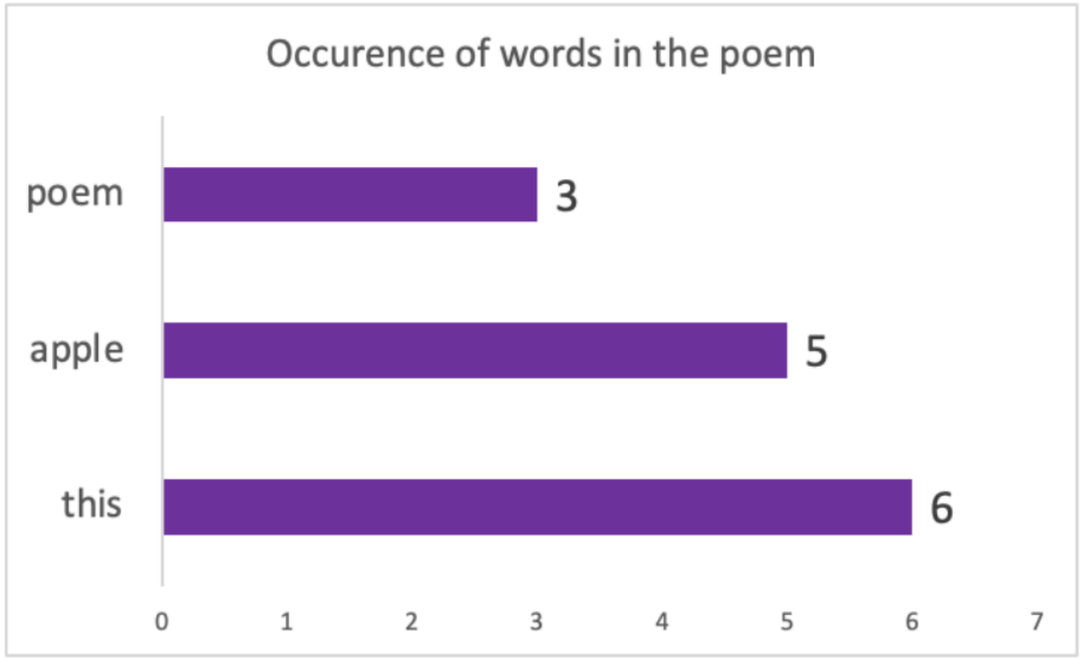


this is not
the leaf of an apple . . . .
or a book . .
. .
this is not a . . this is not an
poem, ripe and ready . . apple waxed poetic
this is not a line that can define what may be true
call it what you like and take a bite if you have a taste
for a would-be-poem in an existential quandary shaped
as a naturally artificial apple, but be careful not to break
your teeth on the concrete, and any way you slice it, this
is rich in fiber, is not red but may be read, is not real but
not entirely unreal, is not still life or a slice of life, make
of it what you will, open it to the core for a metaphor,
dig out seeds for poet trees and juice the ink to make
apple champagne, poetry in a glass, and perhaps
then, when you’re sweetly relaxed, you’ll tell
this not-an-apple, not-a-poem, what it was
and if it tingled, even for a moment, or
if it became some thing or another
because you believed it

Source: ***The Ekphrastic Review***

Chat question

- **How long** did it take you to reach the same conclusion now?



Module completion checklist

Objective	Complete
Identify text cleaning steps for bag-of-words approach	
Implement the text cleaning steps to pre-process documents	

Tokenization: split each document into words

- NLTK's functions operate on **tokens**
- A **token** is the smallest unit of text of interest - in our case, it will be a **word**
- We will use `word_tokenize()` method to split each document into tokens
- Below is a list comprehension that:
  i. Takes each document from the series of document we just created - `df_text`
  ii. Iterates through the each document using `word_tokenize`
  iii. Outputs a large list of tokenized documents

```
# Before we start tokenizing, it is important to check for NAs and drop them
df_text = df_text.dropna().reset_index(drop=True)
# Tokenize each document into a large list of tokenized documents.
df_tokenized = [word_tokenize(df_text[i]) for i in range(0, len(df_text))]
```

Save the first tokenized document

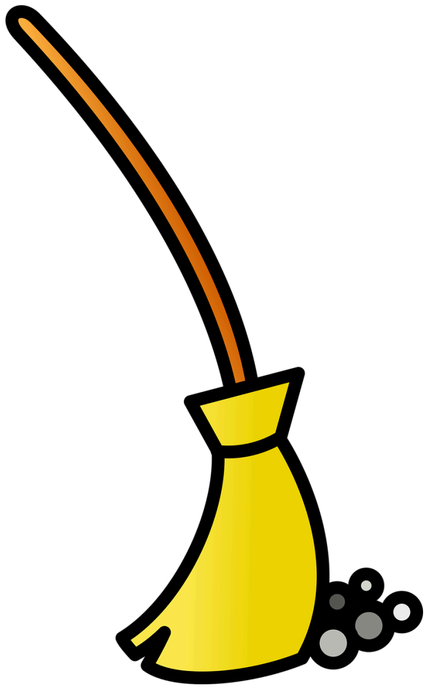
```
# Let's take a look at the first tokenized document
document_words = df_tokenized[0]
print(document_words)
```

```
['Nick', 'Kyrgios', 'started', 'his', 'Brisbane', 'Open', 'title', 'defense', 'with', 'a',
'battling', '7-6', '(', '5', ')', '5-7', '7-6', '(', '5', ')', 'victory', 'over',
'American', 'Ryan', 'Harrison', 'in', 'the', 'opening', 'round', 'on', 'Tuesday', '.']
```

- Let’s test out the cleaning flow on this single document first
- Then we will apply the cleaning steps to all documents in our corpus

“Bag-of-words” analysis: cleaning text flow

1. Convert all characters to lowercase
2. Remove stop words
3. Remove punctuation, numbers, and all other symbols that are not letters of the alphabet
4. Stem words
5. Remove extra white space (if needed)



Convert characters to lowercase

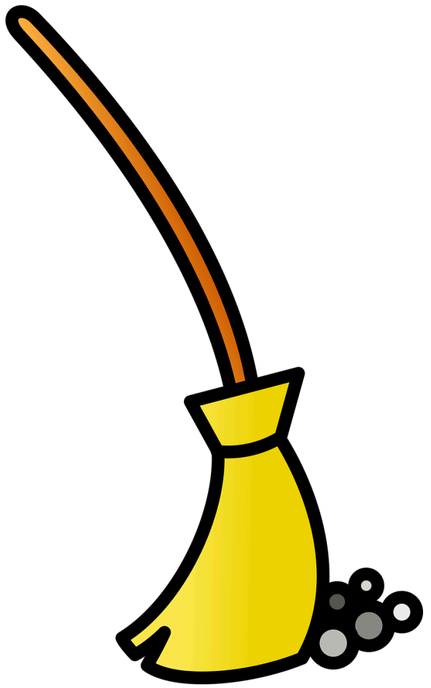
- To convert characters to lowercase, we will use `.lower()` function
  - We call the method like so: `character_string.lower()`
- Since every element of the `document_words` list is a character string, we will convert each word in the document using a **list comprehension**

```
# 1. Convert to lowercase.
document_words = [word.lower() for word in document_words]
print(document_words[:10])
```

```
['nick', 'kyrgios', 'started', 'his', 'brisbane', 'open', 'title', 'defense', 'with', 'a']
```

“Bag-of-words” analysis: cleaning text flow

1. Convert all characters to lowercase
2. Remove stop words
3. Remove punctuation, numbers, and all other symbols that are not letters of the alphabet
4. Stem words
5. Remove extra white space (if needed)



Remove stop words

- In any language, there are words that carry little specific subject-related meaning, but are necessary to bind words into coherent sentences
- Such words are the most frequent and they usually need to be removed, as they create unnecessary noise
- They are called **stop words** and NLTK has a corpus of such words that can be called by using `stopwords` module
  - To get English stop words, we will call `stopwords.words('english')` method:

```
# 2. Remove stop words.
# Get common English stop words.
stop_words = stopwords.words('english')
print(stop_words[:10])
```

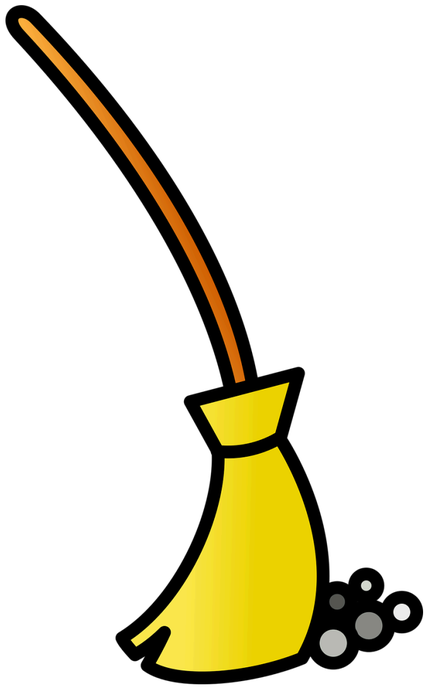
```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]
```

```
# Remove stop words.
document_words = [word for word in document_words if not word in stop_words]
print(document_words[:10])
```

```
['nick', 'kyrgios', 'started', 'brisbane', 'open', 'title', 'defense', 'battling', '7-6',
'(']
```

“Bag-of-words” analysis: cleaning text flow

1. Convert all characters to lowercase
2. Remove stop words
3. Remove punctuation, numbers, and all other symbols that are not letters of the alphabet
4. Stem words
5. Remove extra white space (if needed)



Remove non-alphabetical characters

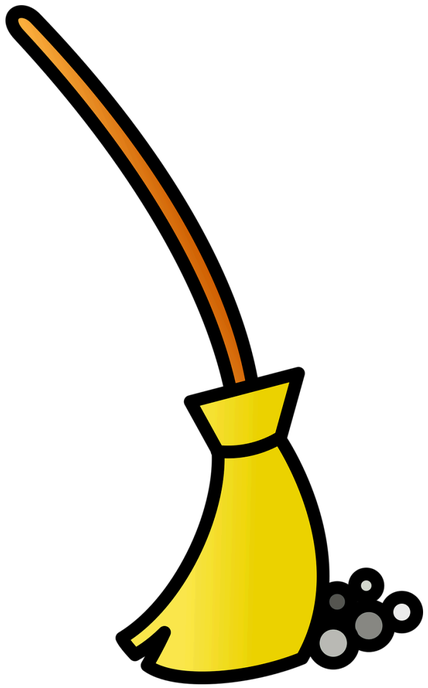
- For text analysis based on “bag-of-words” approach, we only use words
- We remove numbers, punctuation marks or anything other than alphabetical characters
- We will use `.isalpha()` method to check whether the characters are alphabetical or not:
  - We call the method like so: `character_string.isalpha()`
  - Since every element of the `document_words` list is a character string, we will check each token in the document using a **conditional** `if` inside of the **list comprehension**

```
# 3. Remove punctuation and any non-alphabetical characters.
document_words = [word for word in document_words if word.isalpha()]
print(document_words[:10])
```

```
['nick', 'kyrgios', 'started', 'brisbane', 'open', 'title', 'defense', 'battling',
'victory', 'american']
```

“Bag-of-words” analysis: cleaning text flow

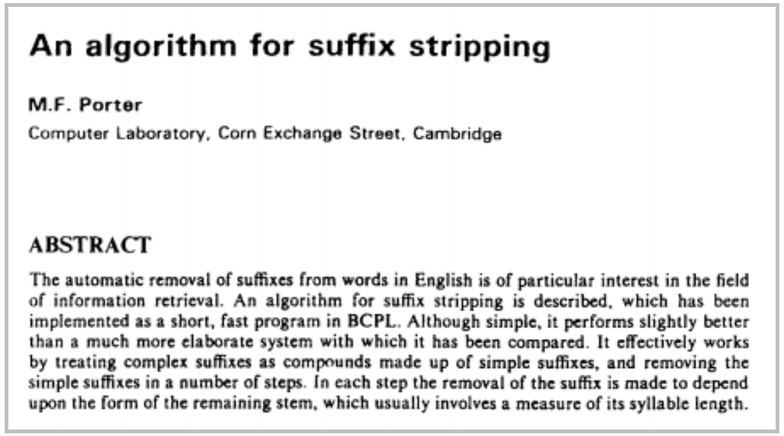
1. Convert all characters to lowercase
2. Remove stop words
3. Remove punctuation, numbers, and all other symbols that are not letters of the alphabet
4. Stem words
5. Remove extra white space (if needed)



Porter stemmer

- **Stemming** reduces words to their root form - it allows us to:
  - treat different forms of the same word as one (i.e. “reads” and “reading” would both be stemmed to “read”)
  - shrink the total number of unique terms, which reduces the noise in data and dimensionality of the data, which we will discuss shortly
- Use the **PorterStemmer** package to stem words in corpus
- **PorterStemmer** is a Python implementation for the most famous stemming algorithm in existence - the Porter stemmer

Porter stemmer (cont'd)



An algorithm for suffix stripping

M.F. Porter
Computer Laboratory, Corn Exchange Street, Cambridge

ABSTRACT

The automatic removal of suffixes from words in English is of particular interest in the field of information retrieval. An algorithm for suffix stripping is described, which has been implemented as a short, fast program in BCPL. Although simple, it performs slightly better than a much more elaborate system with which it has been compared. It effectively works by treating complex suffixes as compounds made up of simple suffixes, and removing the simple suffixes in a number of steps. In each step the removal of the suffix is made to depend upon the form of the remaining stem, which usually involves a measure of its syllable length.

- The algorithm is named after its inventor, computational linguist *Dr. Martin Porter*.
- His paper **"An algorithm for suffix stripping"** is one of the most sited works in Information Retrieval (over **8800** times according to Google Scholar)!

Source: ***Click here (link)***

Stem words

- The `PorterStemmer()` module of NLTK contains a method `.stem()`
- To stem a word, we would simply call `PorterStemmer().stem(word)` for every word in the `document_words` list using a **list comprehension**

```
# 4. Stem words.
document_words = [PorterStemmer().stem(word) for word in document_words]
print(document_words[:10])
```

```
['nick', 'kyrgio', 'start', 'brisban', 'open', 'titl', 'defens', 'battl', 'victori',
'american']
```

Module completion checklist

Objective	Complete
Identify text cleaning steps for bag-of-words approach	✔
Implement the text cleaning steps to pre-process documents	

Implement pre-processing steps on a corpus

- Now that we have successfully implemented text cleaning steps on a single document, we can do that for the entire corpus of documents

```
# Create a list for clean documents.
df_clean = [None] * len(df_tokenized)
```

```
# Create a list of word counts for each clean document.
word_counts_per_document = [None] * len(df_tokenized)
```

```
# Process words in all documents.
for i in range(len(df_tokenized)):
    # 1. Convert to lowercase.
    df_clean[i] = [document.lower() for document in df_tokenized[i]]

    # 2. Remove stop words.
    df_clean[i] = [word for word in df_clean[i] if not word in stop_words]

    # 3. Remove punctuation and any non-alphabetical characters.
    df_clean[i] = [word for word in df_clean[i] if word.isalpha()]

    # 4. Stem words.
    df_clean[i] = [PorterStemmer().stem(word) for word in df_clean[i]]

    # Record the word count per document.
    word_counts_per_document[i] = len(df_clean[i])
```

Inspect results

```
print(df_clean[0][:10])
```

```
['nick', 'kyrgio', 'start', 'brisban', 'open', 'titl', 'defens', 'battl', 'victori',
'american']
```

```
print(df_clean[5][:10])
```

```
['prohibit', 'vacat', 'rental', 'arrang', 'onlin', 'airbnb', 'move', 'closer', 'realiti',
'thursday']
```

```
print(df_clean[10][:10])
```

```
['labor', 'movement', 'press', 'govern', 'get', 'feder', 'employe', 'back', 'work',
'highlight']
```

```
print(df_clean[15][:10])
```

```
['william', 'seed', 'may', 'match', 'simona', 'halep', 'perhap', 'sister', 'venu',
'melbourn']
```

```
print(df_clean[20][:10])
```

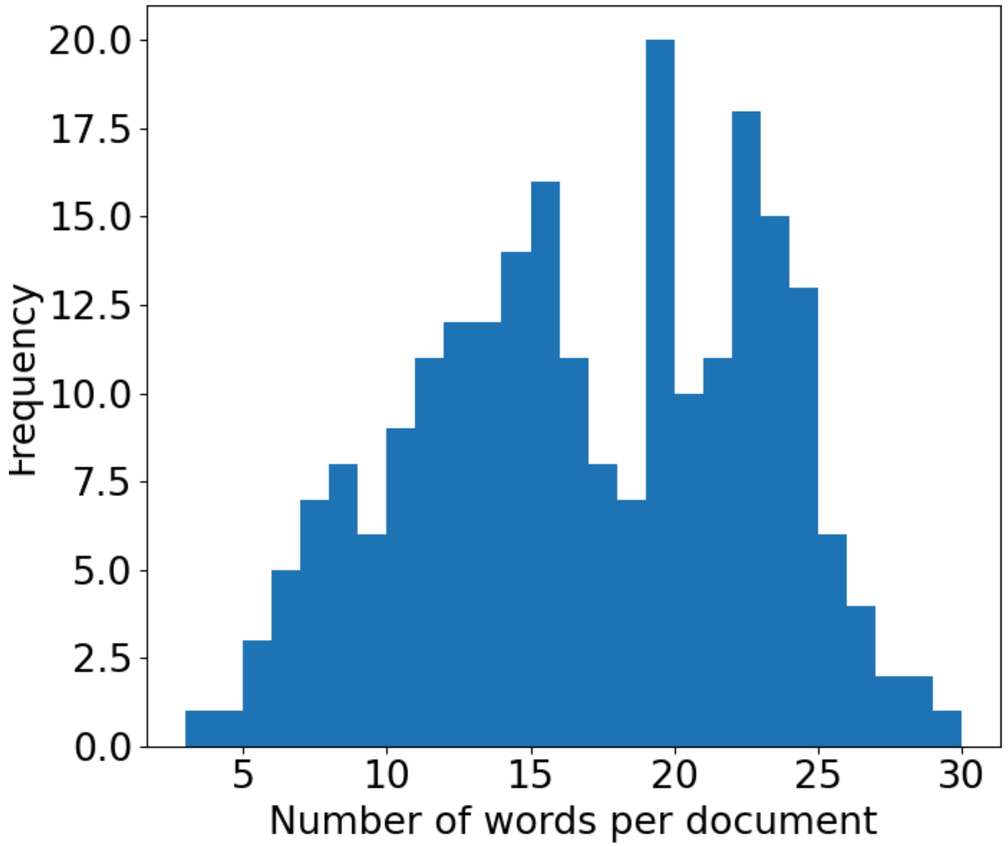
```
['epiphani', 'celebr', 'bring', 'togeth', 'hundr', 'peopl', 'romanian', 'villag',
'pietrosani', 'day']
```

Remove empty and very short documents

```
# Let's take a look at total word counts per
document (for the first 10).
print(word_counts_per_document[:10])
```

```
[15, 22, 27, 17, 19, 12, 5, 11, 23, 13]
```

```
# Plot a histogram for word counts per document,
set bins to number of unique values in the list.
plt.hist(word_counts_per_document, bins =
len(set(word_counts_per_document)))
plt.xlabel('Number of words per document')
plt.ylabel('Frequency')
```



Remove empty and very short documents (cont'd)

- After cleaning, we look to see if there are documents that might not be meaningful anymore
- In this case, we will look for any documents that contain under 5 words

```
# Convert word counts list and documents list to NumPy arrays.
word_counts_array = np.array(word_counts_per_document)
df_array = np.array(df_clean,dtype= object)
print(len(df_array))
```

```
233
```

```
# Find indices of all documents where there are greater than or equal to 5 words.
valid_documents = np.where(word_counts_array >= 5)[0]
print(len(valid_documents))
```

```
231
```

Remove empty and very short documents (cont'd)

- We now only keep all documents over 5 words

```
# Subset the df_array to keep only those where there are at least 5 words.
df_array = df_array[valid_documents]
print(len(df_array))
```

```
231
```

- And convert the array back to a list so we can continue analysis

```
df_clean = df_array.tolist() # Convert the array back to a list.
print(df_clean[:5])
```

```
[['nick', 'kyrgio', 'start', 'brisban', 'open', 'titl', 'defens', 'battl', 'victori',
'american', 'ryan', 'harrison', 'open', 'round', 'tuesday'], ['british', 'polic', 'confirm',
'tuesday', 'treat', 'stab', 'attack', 'injur', 'three', 'peopl', 'manchest', 'victoria',
'train', 'station', 'terrorist', 'investig', 'search', 'address', 'cheetham', 'hill',
'area', 'citi'], ['marcellu', 'wiley', 'still', 'fenc', 'let', 'young', 'son', 'play',
'footbal', 'former', 'nfl', 'defens', 'end', 'fox', 'sport', 'person', 'tell', 'podcaston',
'sport', 'like', 'nfl', 'tri', 'make', 'footbal', 'safer', 'game', 'de'], ['still',
'reckon', 'fallout', 'emmett', 'till', 'paint', 'chasten', 'artist', 'reveal',
'controversi', 'chang', 'even', 'move', 'forward', 'new', 'galleri', 'show'], ['far',
'arik', 'ogunbowal', 'coach', 'muffet', 'mcgraw', 'concern', 'notr', 'dame', 'victori',
'louisvil', 'thursday', 'night', 'anoth', 'atlant', 'coast', 'confer', 'game', 'januari']]
```

- Note: Threshold of max/min words at each instance vary based on subject matter

.join() function

- In the next step, we will be joining the words back together to form complete documents
- **To do this, we need to remember the `.join()` command:**
  - The `.join()` method provides a flexible way to concatenate a string
  - It concatenates each element of an iterable (list, string and tuple) to the string
  - It returns a string concatenated with the elements of an iterable

```
# Here is a simple example of the `.join()` function in action!
numList = ['1', '2', '3', '4']
print(', '.join(numList))
```

```
1, 2, 3, 4
```

Save processed text to file using .join()

```
# Join words in each document into a single character string.
df_clean_list = [' '.join(document) for document in df_clean]
print(df_clean_list[:2])
```

```
['nick kyrgio start brisban open titl defens battl victori american ryan harrison open round
tuesday', 'british polic confirm tuesday treat stab attack injur three peopl manchest
victoria train station terrorist investig search address cheetham hill area citi']
```

```
# Save output file name to a variable.
out_filename = str(data_dir) + "/clean_df.txt"
```

```
# Create a function that takes a list of character strings
# and a name of an output file and writes it into a txt file.
def write_lines(lines, filename):    #<- given lines to write and filename
    joined_lines = '\n'.join(lines)  #<- join lines with line breaks
    file = open( out_filename, 'w')  #<- open write only file
    file.write(joined_lines)         #<- write lines to file
    file.close()                     #<- close connection
```

```
# Write sequences to file.
write_lines(df_clean_list, out_filename)
```

“Bag-of-words” analysis: key elements

What we need	What we have learned
A corpus of documents cleaned and processed in a certain way • All words a converted to lowercase • All punctuation, numbers and special characters are removed • Stopwords are removed • Words are stemmed to their root form	✔
A Document-Term Matrix (DTM): with counts of each word recorded for each document	
A transformed representation of a Document-Term Matrix (i.e. weighted with TF-IDF weights)	

Knowledge check

Module completion checklist

Objective	Complete
Identify text cleaning steps for bag-of-words approach	✔
Implement the text cleaning steps to pre-process documents	✔

Congratulations on completing this module!

You are now ready to try Tasks 4-6 in the Exercise for this topic