DATA SOCIETY®

Text mining - day 1

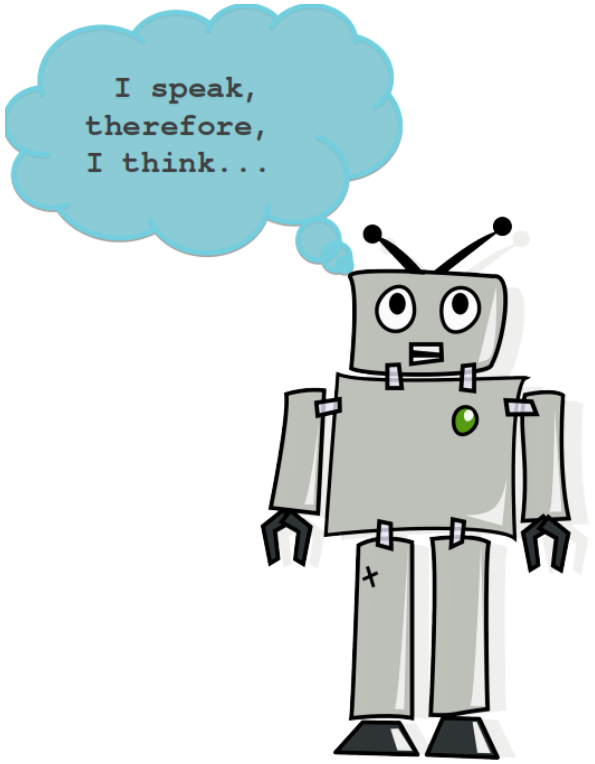
“One should look for what is and not what he thinks should be.”
-Albert Einstein.

Module completion checklist

Objective	Complete
Explain the concept of NLP and how it is used in industry	
Review the tools and packages in Python today to work with text data	
Discuss what a corpus is based on use cases	
Create and inspect a corpus object using NLTK	
Load data into Python using pandas	
Distinguish specific steps to pre-process text for the bag-of-words approach	
Implement steps to pre-process text for the bag-of-words approach	
Create term document matrix and visualize distribution of words	

Text analysis use cases

Field	Use case
Natural language processing	Discover patterns in speech; understand how human language works
Business	Help increase profits through analyzing product reviews; summarize and group reports, catalog documents
Psychology & psychiatry	Analyze sentiments to detect and prevent dangerous social behavior
Network analysis	Augment network analysis to better understand how people interact within a group or react to a certain event
Many more	…



What can text analysis be used for?

Check the text analysis done by Bloomberg on Presidential candidates ***here***

Methodology:

- To conduct this analysis, Bloomberg News collected nearly 24,000 tweets and retweets from official candidate accounts using the Twitter API from January 1 to June 23rd.
- The text of the tweets were classified programmatically using a body of keywords that corresponded to a larger bucket of topics categorized by Bloomberg News.
- The initial keywords were generated by topic modeling the entire corpus of tweets, then supplemented manually with additional keywords.
- No photos, videos or tweets in languages other than English were analyzed

What are NLP, text analysis, and text mining?

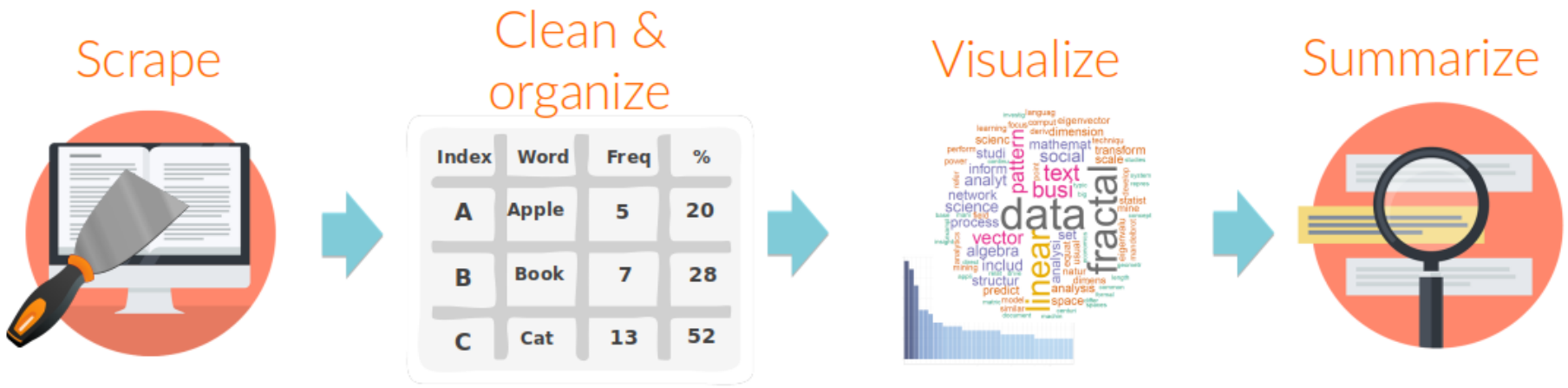
- **Natural Language Processing (NLP)** is a host of methods and disciplines, which reside at the crossroad of data science, computational linguistics, and artificial intelligence
- NLP's main focus is *processing all forms of human language (spoken or written) and getting insightful information out of it* for the purpose of generating human speech, automation of tasks like machine translation, producing predictive models or detecting patterns in human language
- **Text analysis is a part of NLP** and works with *detecting patterns, extracting information, and generating predictive models from written text*
- **Text mining** is a form of information retrieval that is a part of **text analysis**, it concentrates its efforts at *processing large volumes of text and automatically extracting meaningful information out of text*

Why text analysis?

- Most of the information and knowledge in the world is stored as text

Text analysis: flow and basic methods

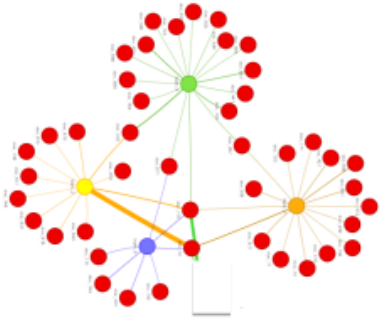
- Text analysis tools range from the simplest to the most advanced
- Here's an example of a flow of text analysis pipeline



Text analysis: more advanced methods

- Once those steps are performed, more complex methods can be used

Classify documents



Extract entities

Analyze sentiment

Datasets for today - class

- Working with **NY Times article snippets**
  - To learn about working with text data using Python
  - To detect patterns in text utilizing text processing and analyzing techniques
  - To apply the above methods to find articles that belong to certain groups (i.e. topics)

Datasets for today - exercises

- Working with **UN agreement titles**
  - To learn about working with text data using Python
  - To detect patterns in text utilizing text processing and analyzing techniques
  - To apply the above methods to find UN agreements that belong to certain groups (i.e. topics)

Module completion checklist

Objective	Complete
Explain the concept of NLP and how it is used in industry	✔
Review the tools and packages in Python today to work with text data	
Discuss what a corpus is based on use cases	
Create and inspect a corpus object using NLTK	
Load data into Python using pandas	
Distinguish specific steps to pre-process text for the bag-of-words approach	
Implement steps to pre-process text for the bag-of-words approach	
Create term document matrix and visualize distribution of words	

Directory settings

- In order to maximize the efficiency of your workflow, you may want to encode your directory structure into `variables`
- Let the `main_dir` be the variable corresponding to your `af-werx` folder

```
# Set `main_dir` to the location of your `af-werx` folder (for Linux).
main_dir = "/home/[username]/af-werx"
```

```
# Set `main_dir` to the location of your `af-werx` folder (for Mac).
main_dir = '/Users/[username]/af-werx'
```

```
# Set `main_dir` to the location of your `af-werx` folder (for Windows).
main_dir = "C:\\Users\\[username]\\af-werx"
```

```
# Make `data_dir` from the `main_dir` and
# remainder of the path to data directory (for Mac).
data_dir = main_dir + "/data"
```

```
# Make `data_dir` from the `main_dir` and
# remainder of the path to data directory (for Windows).
data_dir = main_dir + "\\data"
```

Loading packages

- Load the packages we will be using
- We have used the helper packages so far, the packages that are new are all that are related to text processing

```
# Helper packages.
import os
import pandas as pd
import numpy as np
import pickle
import matplotlib.pyplot as plt
```

```
# Packages with tools for text processing.
from wordcloud import WordCloud
import nltk
import nltk.data
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem.porter import PorterStemmer
from sklearn.feature_extraction.text import CountVectorizer
```

Working directory

- Set working directory to `data_dir`

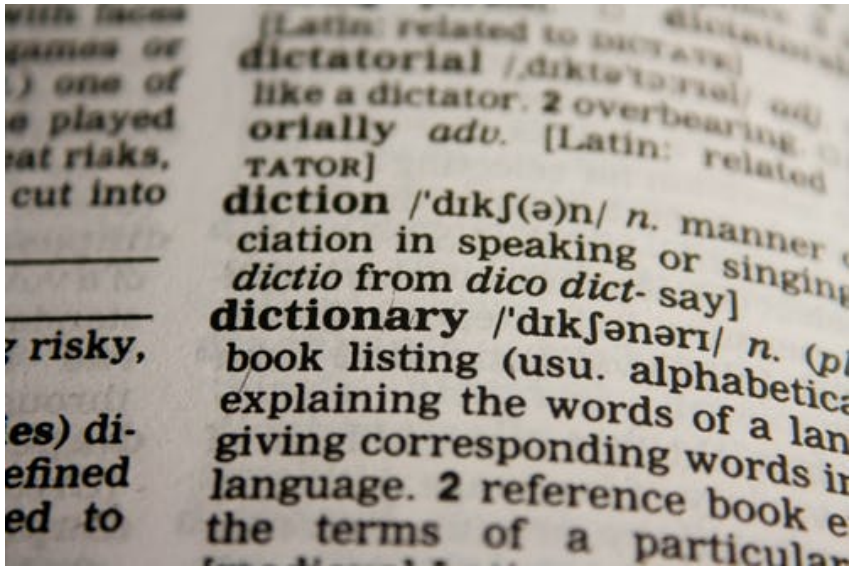
```
# Set working directory.
os.chdir(data_dir)
```

```
# Check working directory.
print(os.getcwd())
```

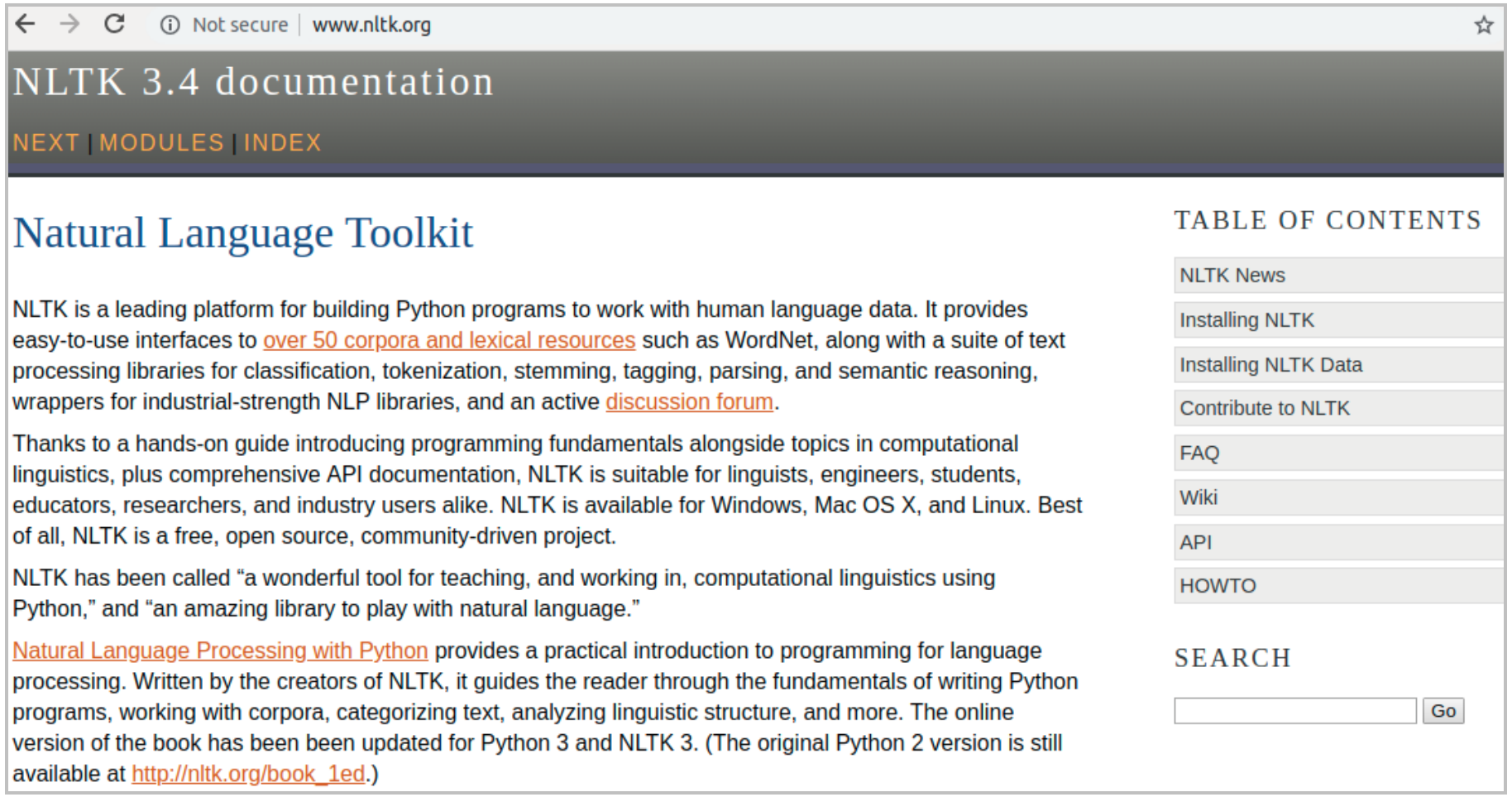
```
/home/[user-name]/af-werx/data
```

A little about NLTK

- Python's **Natural Language Tool Kit** or **NLTK** as it is known is a very powerful platform
- It has been called **"a wonderful tool for teaching, and working in, computational linguistics using Python"** and **"an amazing library to play with natural language"**
- It is **free, open source, easy to use, large community and well documented**
- Today, we will be using NLTK to dive into NLP and text analysis



Processing text in Python with NLTK



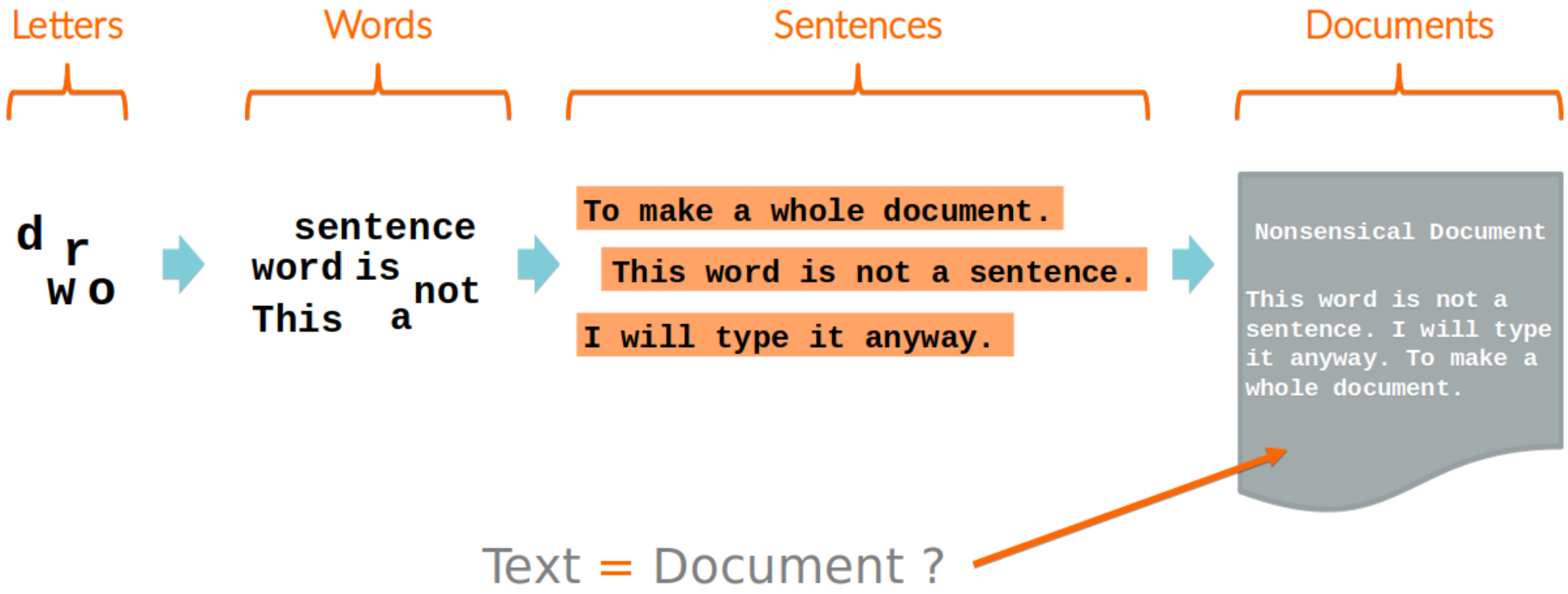
- All NLTK most recent documentation can be found at ***http://www.nltk.org/***

Knowledge check 1

Module completion checklist

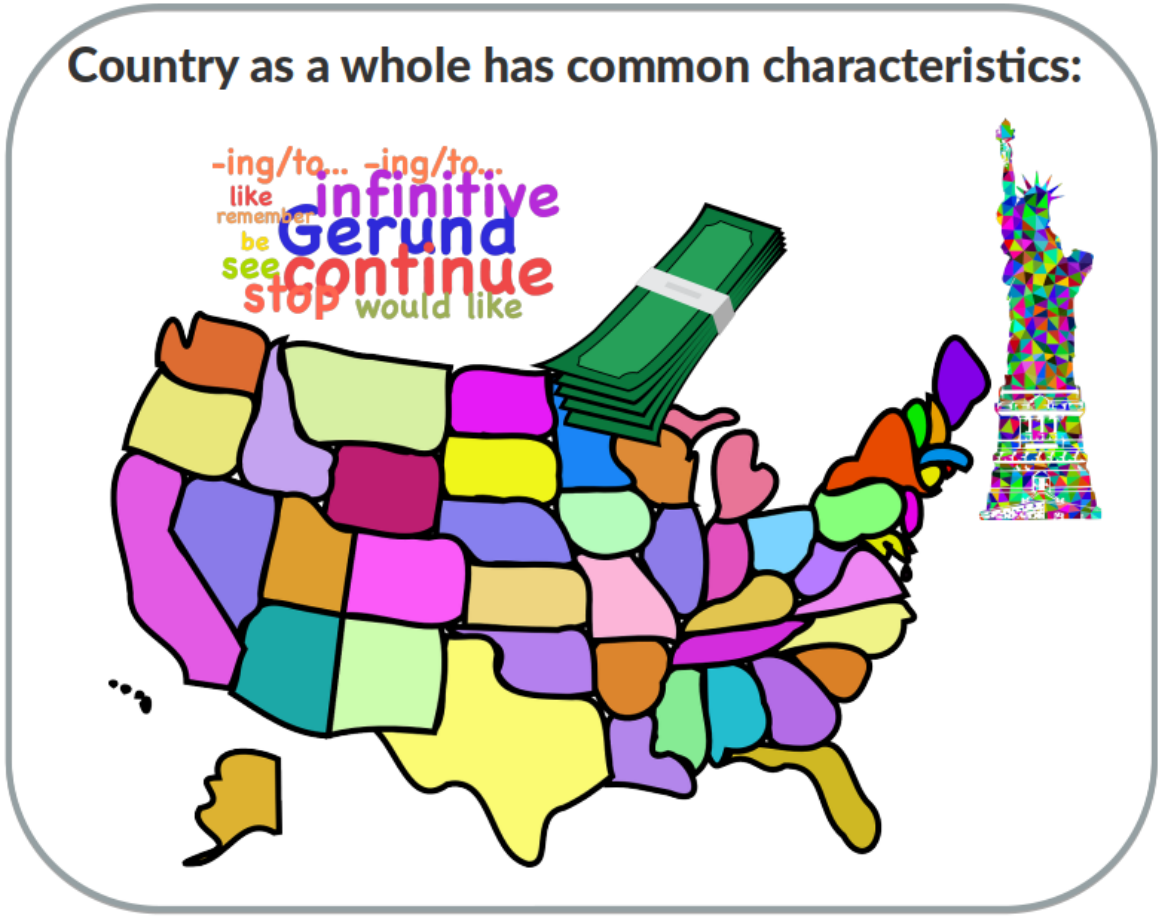
Objective	Complete
Explain the concept of NLP and how it is used in industry	✔
Review the tools and packages in Python today to work with text data	✔
Discuss what a corpus is based on use cases	
Create and inspect a corpus object using NLTK	
Load data into Python using pandas	
Distinguish specific steps to pre-process text for the bag-of-words approach	
Implement steps to pre-process text for the bag-of-words approach	
Create term document matrix and visualize distribution of words	

Units of text data

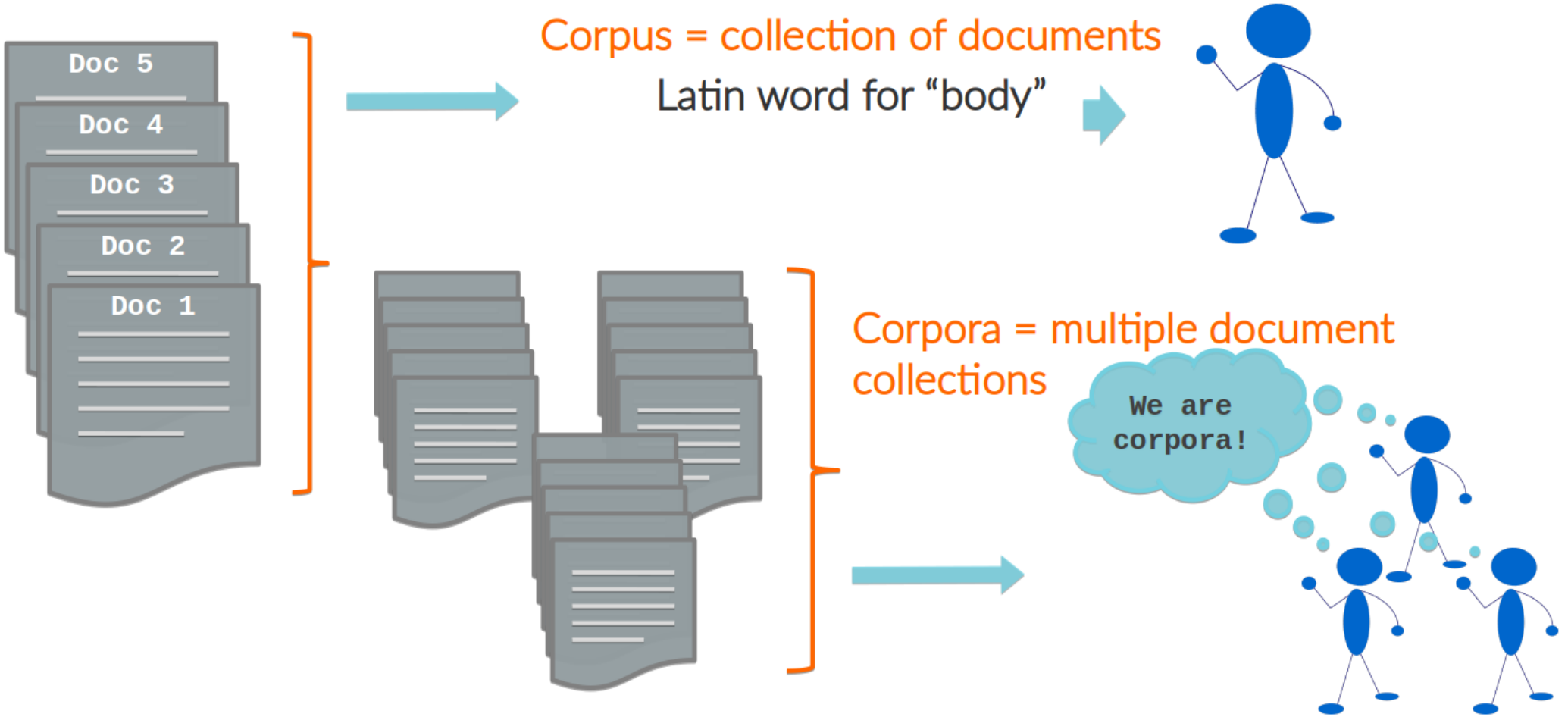


Units of text data: corpus

≈



Units of text data: corpora



Data we will be working with

- The `data` folder inside of your class folder contains all datasets we will be using and generating during this course
- The file that contains the text for analysis is `NYT_article_data.csv`
- The data within this document are **snippets from NYT articles** that we have described already

The New York Times

Units of text: article snippets

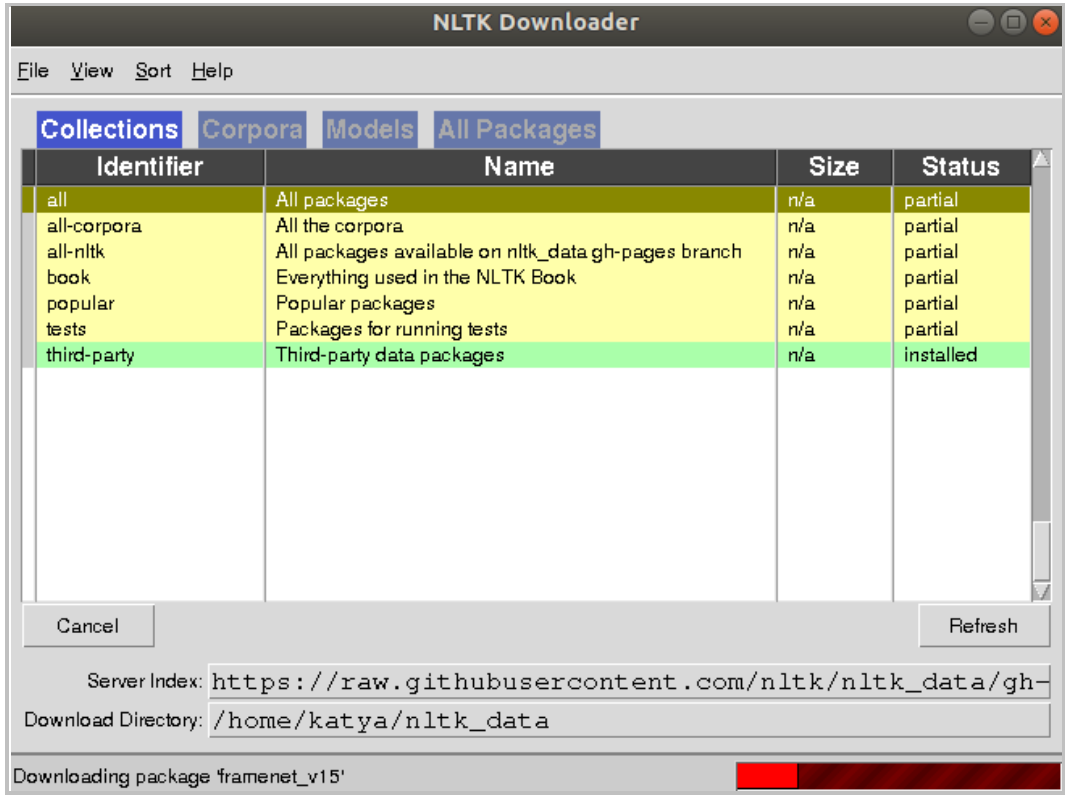
- In this case, we will consider
  - Each **article snippet** as a **document**
  - **All article snippets** together as a **corpus**
- This division of units of text will help us
  - Evaluate each snippet for its subject and potential topic
  - Evaluate all snippets for general word distribution and overall patterns

Module completion checklist

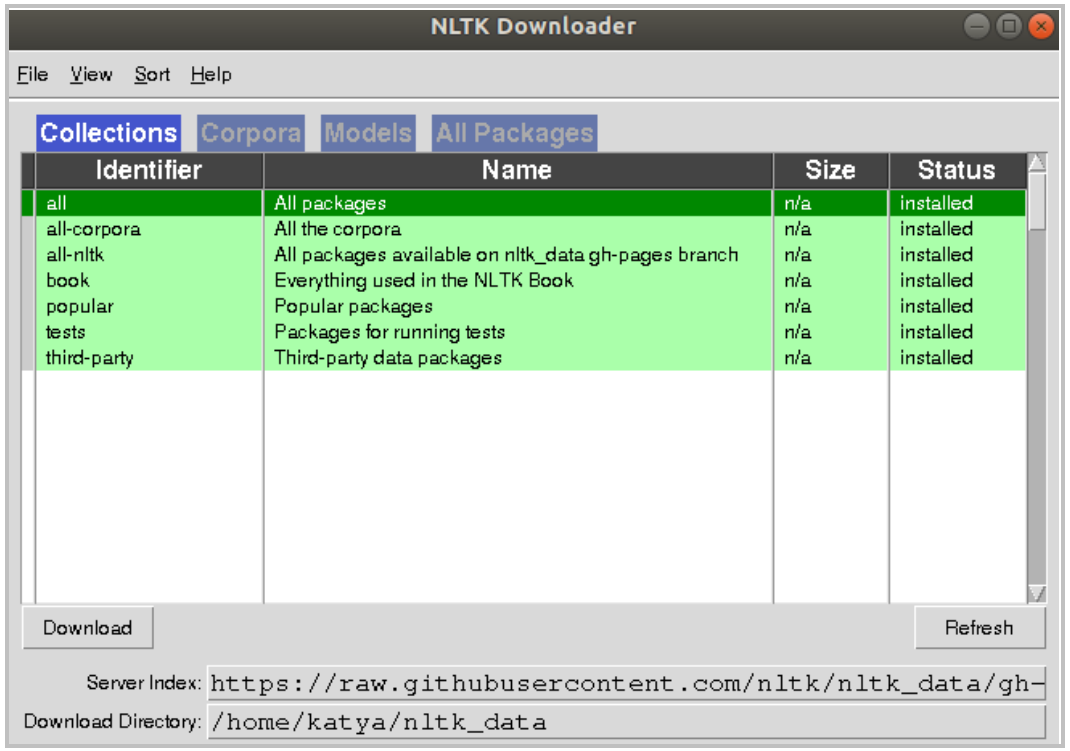
Objective	Complete
Explain the concept of NLP and how it is used in industry	✔
Review the tools and packages in Python today to work with text data	✔
Discuss what a corpus is based on use cases	✔
Create and inspect a corpus object using NLTK	
Load data into Python using pandas	
Distinguish specific steps to pre-process text for the bag-of-words approach	
Implement steps to pre-process text for the bag-of-words approach	
Create term document matrix and visualize distribution of words	

Download NLTK resources

```
# Download resources from NLTK package.
nltk.download()
```



- When you have downloaded all resources, just close this dialogue window



Loading text data

```
# Load the corpus.
NYT = pd.read_csv(data_dir +
'/NYT_article_data.csv')
```

- Let's look at the columns

```
print(NYT.columns)
```

```
Index(['web_url', 'headline', 'snippet',
'word_count', 'source',
       'type_of_material', 'date'],
      dtype='object')
```

- We will be focused on the `snippet` column as of now
- The other columns will be useful as we further our analysis

Look at the columns.

```
# Look at the columns.
print(NYT.head())
```

```
                                              web_url  ...        date
0  https://www.nytimes.com/reuters/2019/01/01/spo...  ...  2019-01-01
1  https://www.nytimes.com/reuters/2019/01/01/spo...  ...  2019-01-01
2  https://www.nytimes.com/reuters/2019/01/01/spo...  ...  2019-01-01
3  https://www.nytimes.com/aponline/2019/01/01/wo...  ...  2019-01-01
4  https://www.nytimes.com/reuters/2019/01/01/spo...  ...  2019-01-01

[5 rows x 7 columns]
```

Creating a list of snippets

```
# Isolate the snippet column.
NYT_snippet = NYT["snippet"]
```

- Look at a sample of the snippets

```
# Look at a sample of the snippets column, 0-20.
print(NYT["snippet"][0:20])
```

```
0     Pakistan's struggling batsmen must find a way ...
1     The National Football League is under the micr...
2     Hitting a hot streak at the right time will be...
3     Pope Francis ushered in the New Year with an o...
4     Chris Froome will not defend his Giro d'Italia...
5     Pakistan's former Prime Minister Nawaz Sharif ...
6     Thousands of demonstrators marched in Hong Kon...
7     Nick Kyrgios started his Brisbane Open title d...
8     British police confirmed on Tuesday they were ...
9     Marcellus Wiley is still on the fence about le...
10    Still reckoning with the fallout from her Emme...
11    As far as Arike Ogunbowale and coach Muffet Mc...
12    A prohibition on "whole-home" vacation rentals...
13           Does contaminated food smell like freedom?
14    There's no end in sight to the partial federal...
15    Bottlenecks for offloading imported fuel are f...
16    Following is reaction to Andy Murray's announc...
17    The labor movement is pressing the government ...
18    House Democrats left a classified briefing wit...
19    The police accused the Spanish tennis player M...
Name: snippet, dtype: object
```

Knowledge check 2

Exercise 1

Module completion checklist

Objective	Complete
Explain the concept of NLP and how it is used in industry	✔
Review the tools and packages in Python today to work with text data	✔
Discuss what a corpus is based on use cases	✔
Create and inspect a corpus object using NLTK	✔
Distinguish specific steps to pre-process text for the bag-of-words approach	
Implement steps to pre-process text for the bag-of-words approach	
Create term document matrix and visualize distribution of words	

Where do we go from here?

- Once the text data is loaded, we need to move to the next stage: **text data preparation**
- Why do we need to prepare the data?
- How do we go about it?

"Bag-of-words" analysis

- How do we quantify and compute on text data? - *We need to translate words into numbers*
- How do we translate words into numbers? - *The simplest solution is to **count** them*
- The analysis of text data based on word counts (a.k.a. frequencies) in documents is called **bag-of-words**

Index	Word	Freq	%
A	Apple	5	20
B	Book	7	28
C	Cat	13	52

"Bag-of-words" analysis: from text to numbers

1. **A document is** treated as **a bag of words** where word position and structure do not matter
2. Text is cleaned until only **stripped down word-roots** remain
3. **Each occurrence** of a word is then counted in each document
4. **Word frequencies** are recorded and arranged into a matrix of words by documents, additional weighting may be applied
5. The **numeric representation** of your text corpus is this **matrix**
6. Document similarity is based on a **similar words** appear in **documents** with similar **meaning**

This is the most basic version of the bag of words data preparation flow!

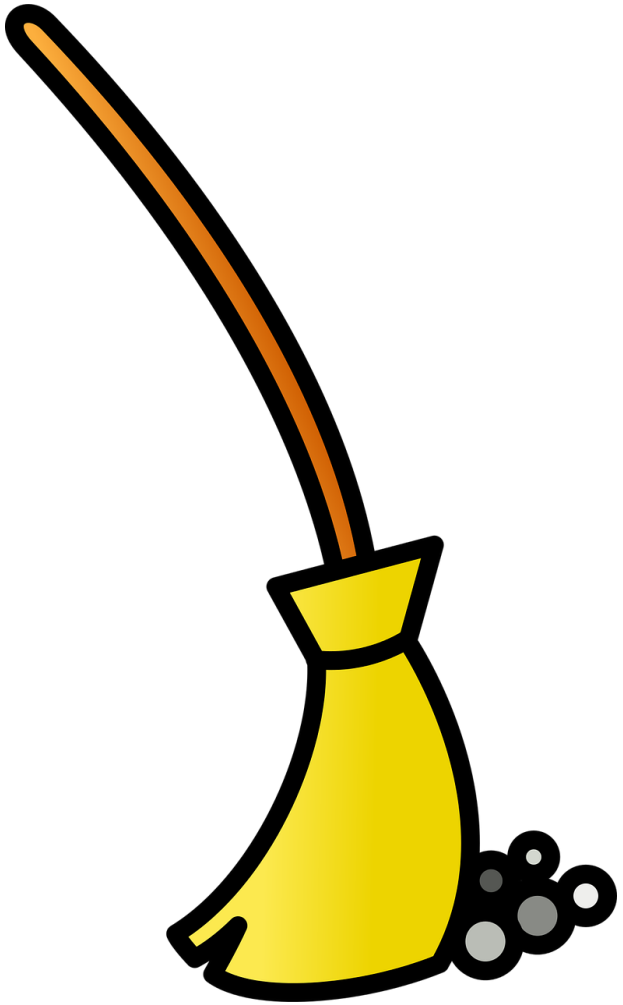
"Bag-of-words" analysis: key elements

What we need	What we have learned
A corpus of documents cleaned and processed in a certain way • All words a converted to lower case • All punctuation, numbers and special characters are removed • Stopwords are removed • Words are stemmed to their root form (and sometimes lemmatized)	
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)	

"Bag-of-words" analysis: cleaning text flow

Text preparation and cleaning is one of the most important steps in text mining and analysis

1. Convert all characters to lower case
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 whitespace (if needed)



Text cleaning steps: order does matter!

✘

1. Remove stop words
2. Convert all characters to lower case

All stop word dictionaries are in lower case - if we do not convert our text to lower case, those stop words that were in upper case will be ignored!

You ≠ you

✔

1. Convert all characters to lower case
2. Remove stop words

When we first unify all words and convert them to lower case, we will be able to catch all instances of stop words!

You → you

you = you

Module completion checklist

Objective	Complete
Explain the concept of NLP and how it is used in industry	✔
Review the tools and packages in Python today to work with text data	✔
Discuss what a corpus is based on use cases	✔
Create and inspect a corpus object using NLTK	✔
Distinguish specific steps to pre-process text for the bag-of-words approach	✔
Implement steps to pre-process text for the bag-of-words approach	
Create term document matrix and visualize distribution of words	

Tokenization: split each snippet 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 snippet into tokens
- Below is a list comprehension that:
  i. Takes each snippet from the list we just created `NYT_snippet`
  ii. Iterates through the each snippet using `word_tokenize`
  iii. Outputs a large list of tokenized snippets

```
# Tokenize each snippet into a large list of tokenized snippets.
NYT_tokenized = [word_tokenize(NYT_snippet[i]) for i in range(0,len(NYT_snippet))]
```

Save the first tokenized snippet

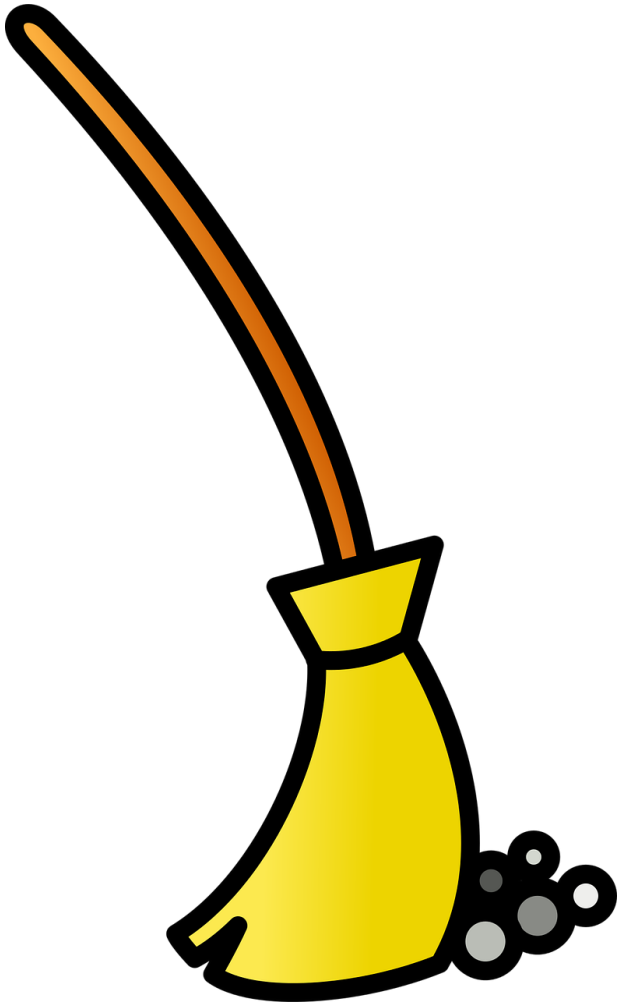
```
# Let's take a look at the first tokenized snippet.
snippet_words = NYT_tokenized[0]
print(snippet_words)
```

```
['Pakistan', "'s", 'struggling', 'batsmen', 'must', 'find', 'a', 'way', 'to', 'handle', 'South',
'Africa', "'s", 'potent', 'pace', 'attack', 'if', 'they', 'are', 'to', 'claw', 'their', 'way', 'back',
'into', 'the', 'three-match', 'series', 'in', 'the', 'second', 'test', 'that', 'starts', 'on', 'what',
'is', 'likely', 'to', 'be', 'a', 'lively', 'Newlands', 'wicket', 'on', 'Thursday', '.']
```

- Let's test out the cleaning flow on this single snippet first
- Then we will apply the cleaning steps to all snippets in our snippet corpus

"Bag-of-words" analysis: cleaning text flow

1. Convert all characters to lower case
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 whitespace (if needed)



Convert characters to lower case

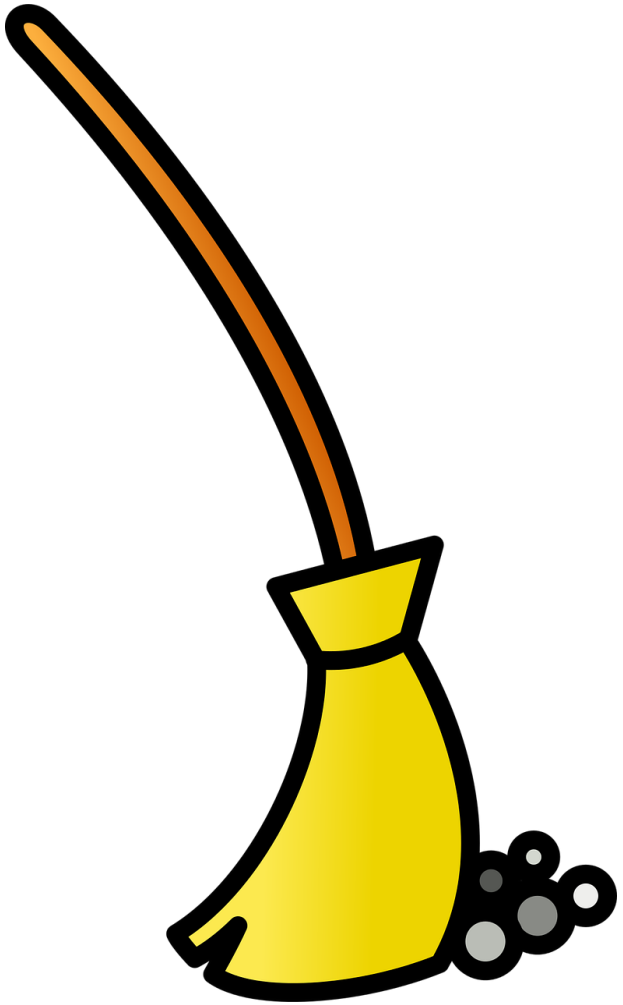
- To convert characters to lower case, we will use `.lower()` function
  - We call the method like so: `character_string.lower()`
- Since every element of the `snippet_words` list is a character string, we will convert each word in the snippet using a **list comprehension**

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

```
['pakistan', "'s", 'struggling', 'batsmen', 'must', 'find', 'a', 'way', 'to', 'handle']
```

"Bag-of-words" analysis: cleaning text flow

1. Convert all characters to lower case
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 whitespace (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 together
- Such words are the most frequent and they usually need to be taken out, 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 stopwords.
# Get common English stop words.
stop_words = stopwords.words('english')
print(stop_words[:10])
```

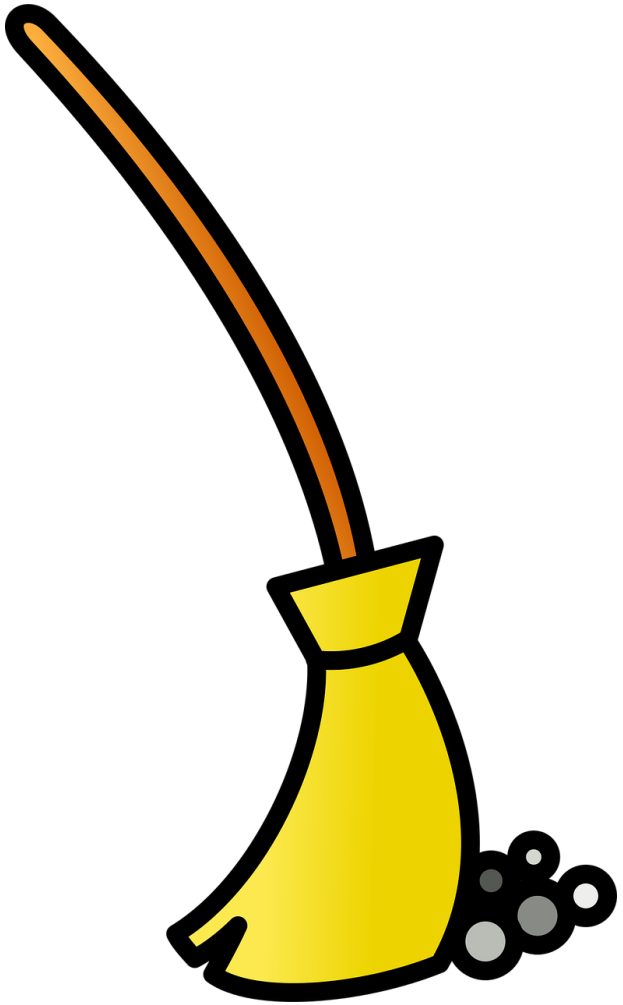
```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]
```

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

```
['pakistan', "'s", 'struggling', 'batsmen', 'must', 'find', 'way', 'handle', 'south', 'africa']
```

"Bag-of-words" analysis: cleaning text flow

1. Convert all characters to lower case
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 whitespace (if needed)



Remove non-alphabetical characters

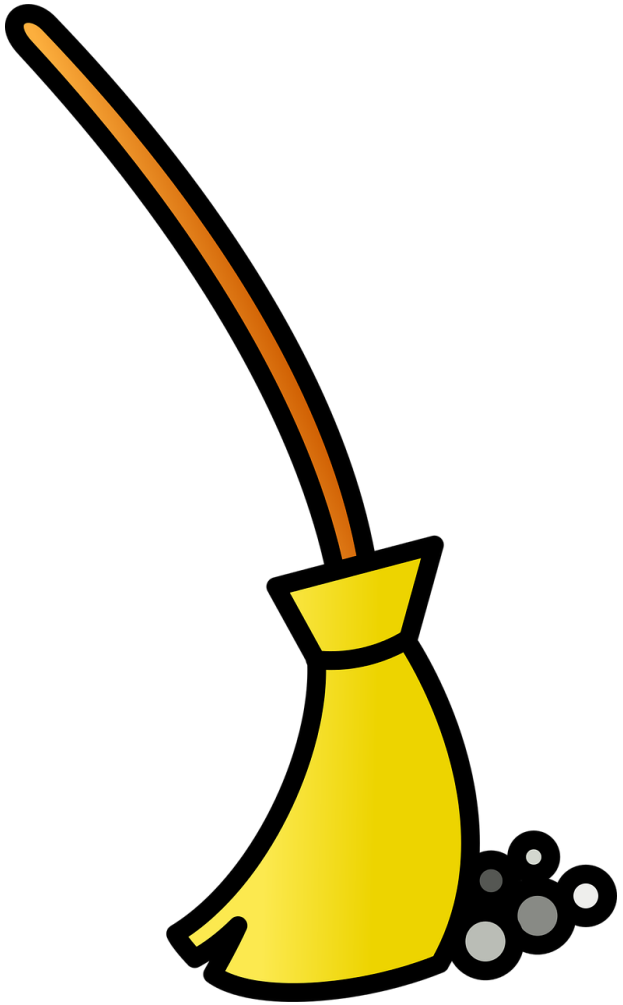
- For text analysis based on "bag-of-words" approach, we only use words
- We remove numbers, punctuation or anything other than alphabetical characters
- We will use `.isalpha()` method to check whether the characters are alphabetical or not
  - We call the method like so: `character_string.isalpha()`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

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

```
['pakistan', 'struggling', 'batsmen', 'must', 'find', 'way', 'handle', 'south', 'africa', 'potent']
```

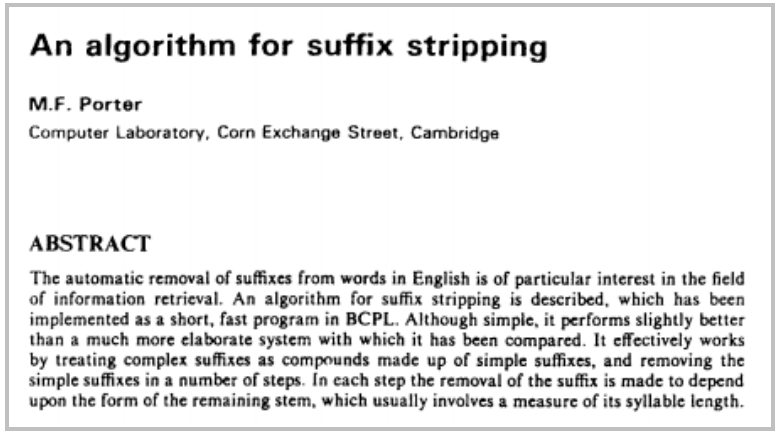
"Bag-of-words" analysis: cleaning text flow

1. Convert all characters to lower case
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 whitespace (if needed)



Porter stemmer

- **Stemming reduces words to their root** form - it allows us to
  - treat different forms of the same word as one
  - 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



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 a computational linguist *Dr. Martin Porter* who invented it
- 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: http://www.cs.odu.edu/~jbollen/IR04/readings/readings5.pdf

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 `snippet_words` list using a **list comprehension**

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

```
['pakistan', 'struggl', 'batsmen', 'must', 'find', 'way', 'handl', 'south', 'africa', 'potent']
```

Implementing pre-processing steps on a corpus

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

```
# Create a list for clean snippets.
NYT_clean = [None] * len(NYT_tokenized)
```

```
# Create a list of word counts for each clean snippet.
word_counts_per_snippet = [None] * len(NYT_tokenized)
```

```
# Process words in all snippets.
for i in range(len(NYT_tokenized)):
    # 1. Convert to lower case.
    NYT_clean[i] = [snippet.lower() for snippet in NYT_tokenized[i]]

    # 2. Remove stopwords.
    NYT_clean[i] = [word for word in NYT_clean[i] if not word in stop_words]

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

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

    # Record the word count per snippet.
    word_counts_per_snippet[i] = len(NYT_clean[i])
```

Inspect results

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

```
['pakistan', 'struggl', 'batsmen', 'must', 'find', 'way', 'handl', 'south', 'africa', 'potent']
```

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

```
['pakistan', 'former', 'prime', 'minist', 'nawaz', 'sharif', 'appeal', 'convict', 'prison', 'sentenc']
```

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

```
['still', 'reckon', 'fallout', 'emmett', 'till', 'paint', 'chasten', 'artist', 'reveal', 'controversi']
```

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

```
['bottleneck', 'offload', 'import', 'fuel', 'form', 'mexican', 'oil', 'port', 'follow', 'govern']
```

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

```
['taiwanes', 'presid', 'tsai', 'appoint', 'close', 'polit', 'alli', 'premier', 'cabinet', 'reshuffl']
```

Removing empty and very short snippets

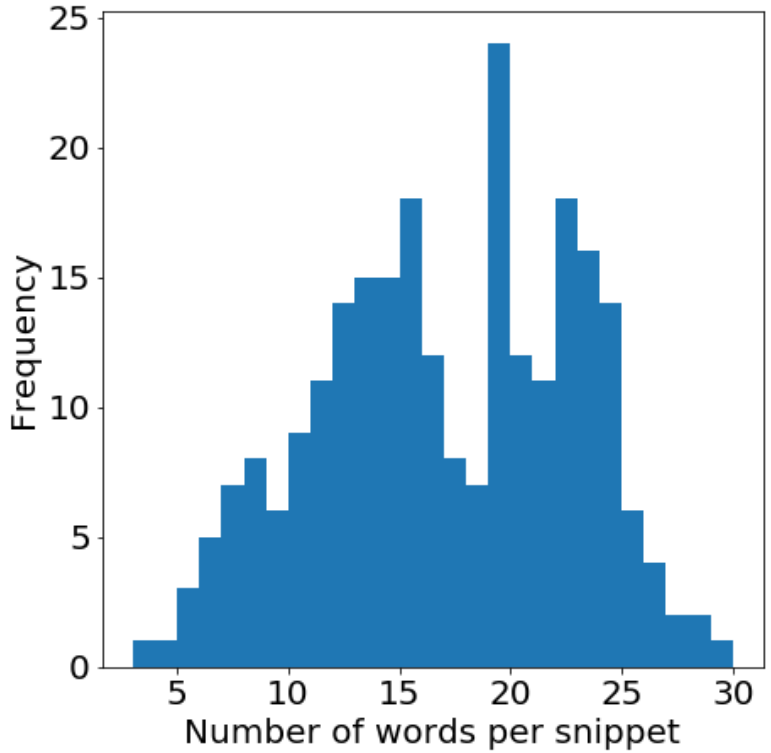
```
# Let's take a look at total word counts per
snippet (for the first 10).
print(word_counts_per_snippet[:10])
```

```
[24, 12, 19, 20, 19, 15, 23, 15, 22, 27]
```

```
# Plot a histogram for word counts per snippet,
set bins to num of unique values in the list.
plt.hist(word_counts_per_snippet, bins =
len(set(word_counts_per_snippet)))
```

```
(array([ 1.,  1.,  3.,  5.,  7.,  8.,  6.,  9.,
11., 14., 15., 15., 18.,
       12.,  8.,  7., 24., 12., 11., 18., 16.,
14.,  6.,  4.,  2.,  2.,
        1.]), array([ 3.,  4.,  5.,  6.,  7.,
8.,  9., 10., 11., 12., 13., 14., 15.,
       16., 17., 18., 19., 20., 21., 22., 23.,
24., 25., 26., 27., 28.,
       29., 30.]), <a list of 27 Patch objects>)
```

```
plt.xlabel('Number of words per snippet')
plt.ylabel('Frequency')
```



Removing empty and very short snippets (cont'd)

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

```
# Convert word counts list and snippets list to numpy arrays.
word_counts_array = np.array(word_counts_per_snippet)
NYT_array = np.array(NYT_clean)
print(len(NYT_array))
```

```
250
```

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

```
248
```

Removing empty and very short snippets (cont'd)

- We now only keep all snippets over 5 words

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

```
248
```

Removing empty and very short snippets (cont'd)

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

```
# Convert the array back to a list.
NYT_clean = NYT_array.tolist()
print(NYT_clean[:10])
```

```
[['pakistan', 'struggl', 'batsmen', 'must', 'find', 'way', 'handl', 'south', 'africa', 'potent',
'pace', 'attack', 'claw', 'way', 'back', 'seri', 'second', 'test', 'start', 'like', 'live', 'newland',
'wicket', 'thursday'], ['nation', 'footbal', 'leagu', 'microscop', 'lack', 'minor', 'head', 'coach',
'recent', 'slew', 'fire', 'leagu'], ['hit', 'hot', 'streak', 'right', 'time', 'goal', 'golf', 'top',
'male', 'profession', 'year', 'new', 'calendar', 'cram', 'major', 'championship', 'super', 'busi',
'stretch'], ['pope', 'franci', 'usher', 'new', 'year', 'ode', 'motherhood', 'tuesday', 'remind',
'faith', 'mother', 'exampl', 'embrac', 'best', 'antidot', 'today', 'disjoint', 'world', 'solitud',
'miseri'], ['chri', 'froom', 'defend', 'giro', 'titl', 'year', 'choos', 'focu', 'win', 'fifth', 'tour',
'de', 'franc', 'crown', 'instead', 'team', 'sky', 'announc', 'tuesday'], ['pakistan', 'former',
'prime', 'minist', 'nawaz', 'sharif', 'appeal', 'convict', 'prison', 'sentenc', 'hand', 'islamabad',
'tribun', 'last', 'week'], ['thousand', 'demonstr', 'march', 'hong', 'kong', 'tuesday', 'demand',
'full', 'democraci', 'fundament', 'right', 'even', 'independ', 'china', 'face', 'mani', 'see', 'mark',
'clampdown', 'communist', 'parti', 'local', 'freedom'], ['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']]
```

- **The threshold of max/min words for each instance varies based on subject matter**

.join() function

- In the next step, we will be joining the words back together to form complete snippets
- **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 snippet into a single character string.
NYT_clean_list = [' '.join(snippet) for snippet in NYT_clean]
print(NYT_clean_list[:5])
```

```
['pakistan struggl batsmen must find way handl south africa potent pace attack claw way back seri
second test start like live newland wicket thursday', 'nation footbal leagu microscop lack minor head
coach recent slew fire leagu', 'hit hot streak right time goal golf top male profession year new
calendar cram major championship super busi stretch', 'pope franci usher new year ode motherhood
tuesday remind faith mother exampl embrac best antidot today disjoint world solitud miseri', 'chri
froom defend giro titl year choos focu win fifth tour de franc crown instead team sky announc tuesday']
```

```
# Save output file name to a variable.
out_filename = data_dir + "/clean_NYT.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(NYT_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 lower case • 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)	

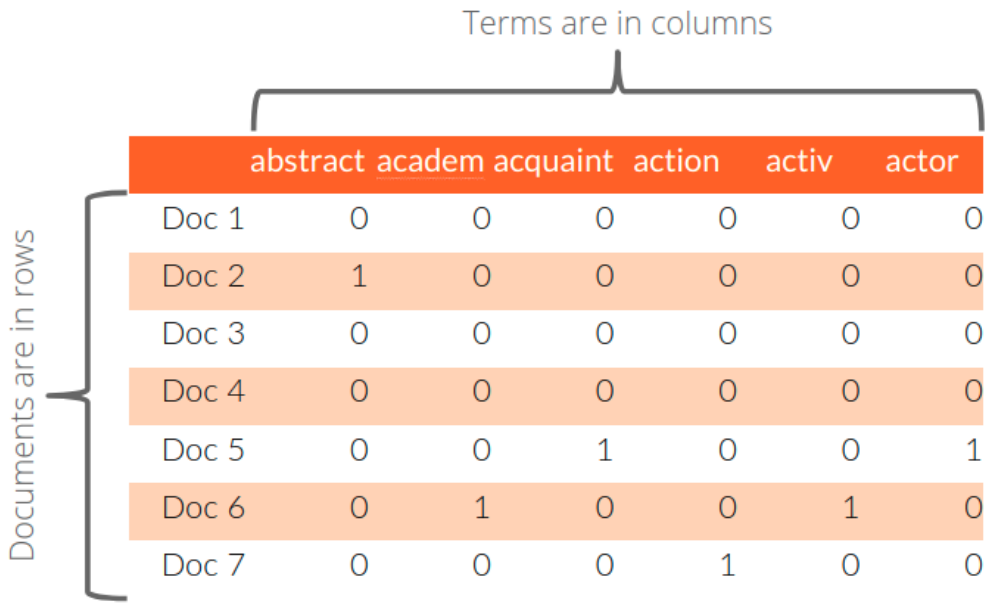
Knowledge check 3

Exercise 2

Module completion checklist

Objective	Complete
Explain the concept of NLP and how it is used in industry	✔
Review the tools and packages in Python today to work with text data	✔
Discuss what a corpus is based on use cases	✔
Create and inspect a corpus object using NLTK	✔
Distinguish specific steps to pre-process text for the bag-of-words approach	✔
Implement steps to pre-process text for the bag-of-words approach	✔
Create term document matrix and visualize distribution of words	

What is a Document-Term Matrix (DTM)?



	abstract	academ	acquaint	action	activ	actor
Doc 1	0	0	0	0	0	0
Doc 2	1	0	0	0	0	0
Doc 3	0	0	0	0	0	0
Doc 4	0	0	0	0	0	0
Doc 5	0	0	1	0	0	1
Doc 6	0	1	0	0	1	0
Doc 7	0	0	0	1	0	0

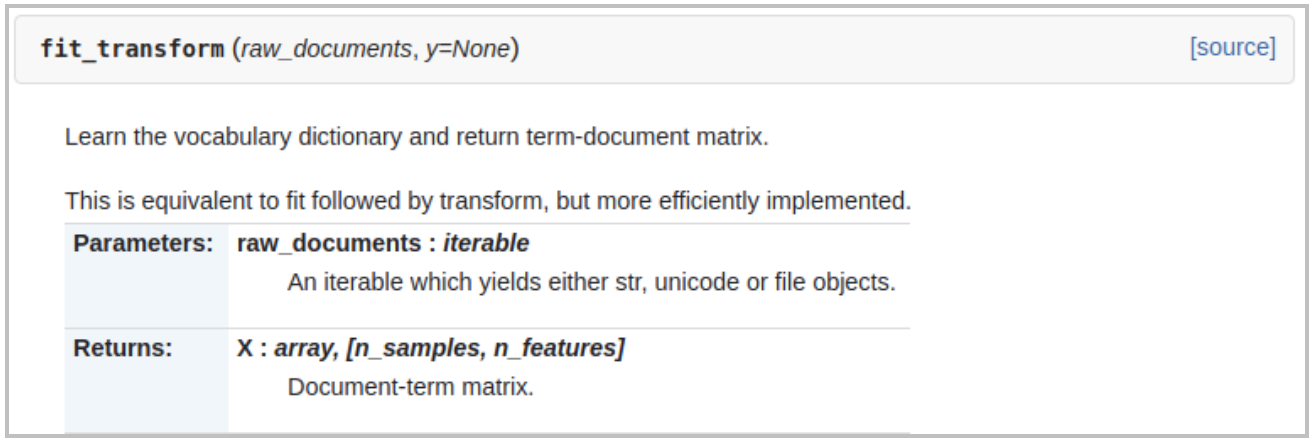
- **Document-term matrix** is simply a matrix of unique words counted in each document:
  - documents are arranged in the rows
  - unique terms are arranged in columns
- The corpus **vocabulary** consists of all of the unique terms (i.e. column names of DTM) and their total counts across all documents (i.e. column sums)

Create DTM with CountVectorizer

- To create a Document-Term matrix, we will use `CountVectorizer` from `scikit-learn` library's `feature_extraction` module for working with text
- `scikit-learn` is another very powerful platform in Python, it is used heavily for machine learning, you can find complete documentation ***here***

Create DTM with CountVectorizer (cont'd)

- It takes a `list` of character strings that represents the documents (i.e. our snippets) as the main argument passed to its `fit_transform()` method:
  - `.fit_transform(list_of_documents)`
- It returns a 2D array (i.e. a matrix) with documents in rows and terms in columns - the **DTM**



fit_transform (*raw_documents*, *y=None*) [source]

Learn the vocabulary dictionary and return term-document matrix.

This is equivalent to fit followed by transform, but more efficiently implemented.

Parameters:	**raw_documents : *iterable*** An iterable which yields either str, unicode or file objects.
Returns:	**X : *array, [n_samples, n_features]*** Document-term matrix.

- For more information on this module, see ***scikit-learn's official documentation on this module***

Create a DTM

```
# Initialize `CountVectorizer`.
vec = CountVectorizer()
```

```
# Transform the list of snippets into DTM.
X = vec.fit_transform(NYT_clean_list)
print(X.toarray()) #<- show output as a matrix
```

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 1 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

- To get a list of names of columns (i.e. the unique terms in our corpus), we can use a utility method `.get_feature_names()`

```
print(vec.get_feature_names()[:10])
```

```
['abduct', 'abl', 'abo', 'absente', 'abus', 'academ', 'accept', 'access', 'accessori', 'accommod']
```

Create a DTM (cont'd)

- Let's convert the matrix into a dataframe, where rows are IDs of the snippets and columns are unique words that appear in those snippets

```
# Convert the matrix into a pandas dataframe for easier manipulation.
DTM = pd.DataFrame(X.toarray(), columns = vec.get_feature_names())
print(DTM.head())
```

```
   abduct  abl  abo  absente  abus  ...  york  young  yuan  zimbabw  zykera
0       0    0    0        0     0  ...     0      0     0        0       0
1       0    0    0        0     0  ...     0      0     0        0       0
2       0    0    0        0     0  ...     0      0     0        0       0
3       0    0    0        0     0  ...     0      0     0        0       0
4       0    0    0        0     0  ...     0      0     0        0       0

[5 rows x 1921 columns]
```

DTM to dictionary of total word counts

- NLTK word frequency visualization functions work with dictionaries
- **Before we convert our DTM to a dictionary**, let's create a convenience function that sorts all words in descending order by counts and displays the first `n` entries
- We use `lambda` within the function, which is an anonymous function

```
# Create a convenience function that sorts and looks at first n-entries in the dictionary.
def HeadDict(dict_x, n):
    # Get items from the dictionary and sort them by
    # value key in descending (i.e. reverse) order
    sorted_x = sorted(dict_x.items(),
                      reverse = True,
                      key = lambda kv: kv[1])

    # Convert sorted dictionary to a list.
    dict_x_list = list(sorted_x)

    # Return the first `n` values from the dictionary only.
    return(dict(dict_x_list[:n]))
```

- **Note: lambda is great when you simply want to insert a function inline because you don't have to define it** - it's a no-name shorthand function

DTM to dictionary of total word counts (cont'd)

```
# Sum frequencies of each word in all documents.
DTM.sum(axis = 0).head()
```

```
abduct     1
abl        1
abo        1
absente    1
abus       2
dtype: int64
```

```
# Save series as a dictionary.
corpus_freq_dist = DTM.sum(axis = 0).to_dict()
```

```
# Glance at the frequencies.
print(HeadDict(corpus_freq_dist, 6))
```

```
{'said': 42, 'new': 39, 'year': 32, 'presid': 29, 'friday': 22, 'govern': 22}
```

"Bag-of-words" analysis: key elements

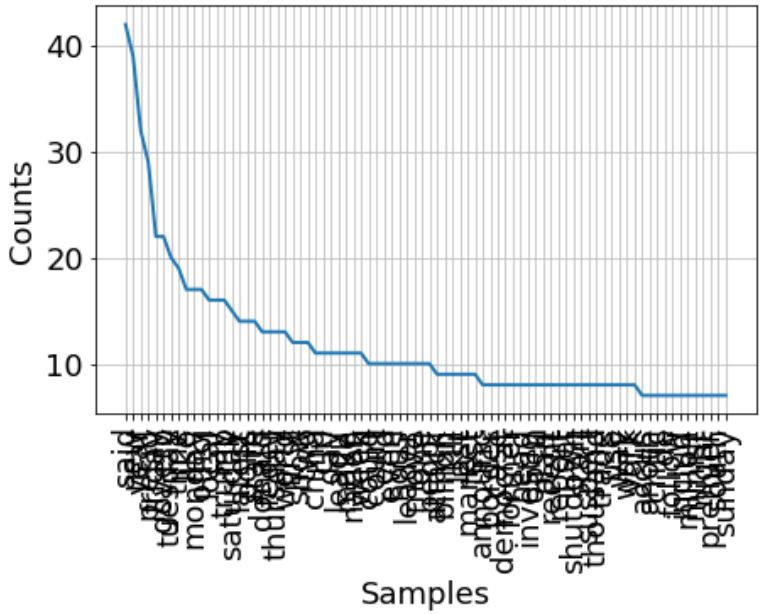
- We have one more step remaining to learn, we will cover this next!

What we need	What we have learned
A corpus of documents cleaned and processed in a certain way • All words a converted to lower case • 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)	

Plot distribution of words in snippet corpus

```
# Save as a FreqDist object native to nltk.
corpus_freq_dist = nltk.FreqDist(corpus_freq_dist)
```

```
# Plot distribution for the entire corpus.
plt.figure(figsize = (16, 7))
corpus_freq_dist.plot(80)
```



Visualizing word counts with word clouds

```
# Construct a word cloud from corpus.
wordcloud = WordCloud(max_font_size = 40,
background_color = "white")
wordcloud = wordcloud.generate('
'.join(NYT_clean_list))

# Plot the cloud using matplotlib.
plt.figure()
plt.imshow(wordcloud, interpolation =
"bilinear")
plt.axis("off")
```

```
(-0.5, 399.5, 199.5, -0.5)
```

```
plt.show()
```

Pickle - what?

- The last thing we will do today is to save our objects that we will need for the next class
- How do we do that? We use a function in Python called `pickle`
- It is similar to **flattening** a file
  - **Pickle/saving: a Python object is converted into a byte stream**
  - **Unpickle/loading: the inverse operation where a byte stream is converted back into an object**

Save results as a pickle

- Pickle all generated data for next steps

```
pickle.dump(DTM, open('DTM.sav', 'wb'))
pickle.dump(X, open('DTM_matrix.sav', 'wb'))
pickle.dump(NYT_clean, open('NYT_clean.sav', 'wb'))
pickle.dump(NYT_clean_list, open('NYT_clean_list.sav', 'wb'))
```

Knowledge check 4

Exercise 3

Module completion checklist

Objective	Complete
Explain the concept of NLP and how it is used in industry	✔
Review the tools and packages in Python today to work with text data	✔
Discuss what a corpus is based on use cases	✔
Create and inspect a corpus object using NLTK	✔
Distinguish specific steps to pre-process text for the bag-of-words approach	✔
Implement steps to pre-process text for the bag-of-words approach	✔
Create term document matrix and visualize distribution of words	✔

Workshop!

- Workshops are to be completed in the afternoon either with a dataset for a capstone project or with another dataset of your choosing
- Make sure to annotate and comment your code so that it is easy for others to understand what you are doing
- This is an exploratory exercise to get you comfortable with the content we discussed today
- Today you will:
  - Get a text data of your choice
  - Load the data, analyze and preprocess it
  - Convert the corpus to bag of words and save the pickle file for next class

This completes our module

Congratulations!