# DATA SOCIETY®

## Introduction to text mining - part 1

*"One should look for what is and not what he thinks should be."*
*-Albert Einstein.*

# Who we are

- Data Society's mission is to **integrate Big Data and machine learning best practices across entire teams** and empower professionals to identify new insights
- We provide:
  - High-quality data science training programs
  - Customized executive workshops
  - Custom software solutions and consulting services
- Since 2014, we've worked with thousands of professionals to make their data work for them

# Using Zoom

**Raise your hand if you can hear me OK.**

- From the toolbar (probably on the bottom of your screen), ***select the button marked "Reactions" and choose the hand***

**In the chat box, tell everyone what you see out the nearest window.**

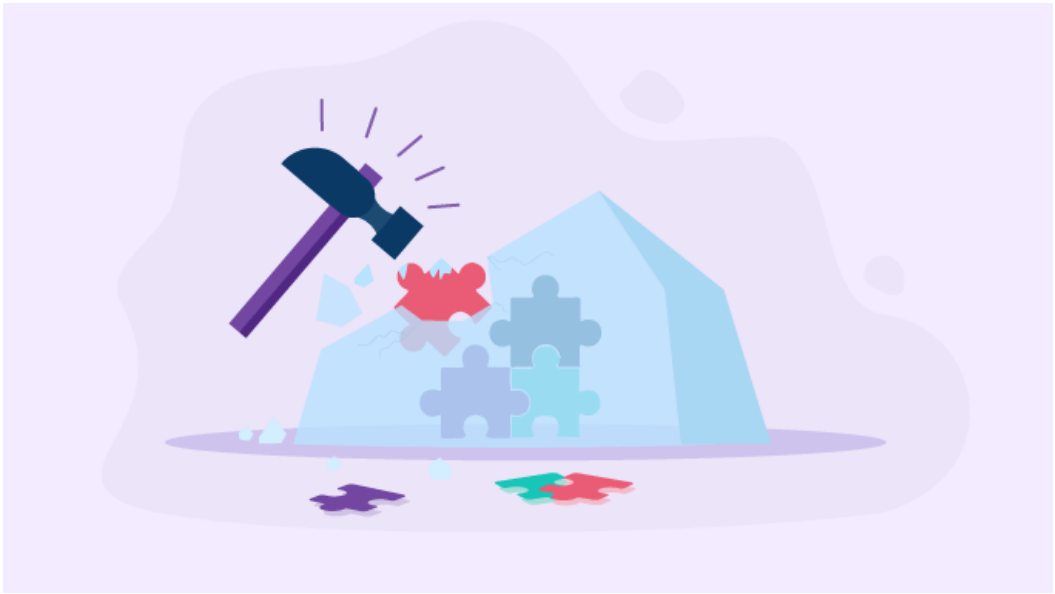
- From the toolbar (probably on the bottom of your screen), ***select the button marked "Chat."*** The chat box should appear. On smaller screens, the "Chat" button may be hiding under the "More" menu.

# Best practices for virtual classes

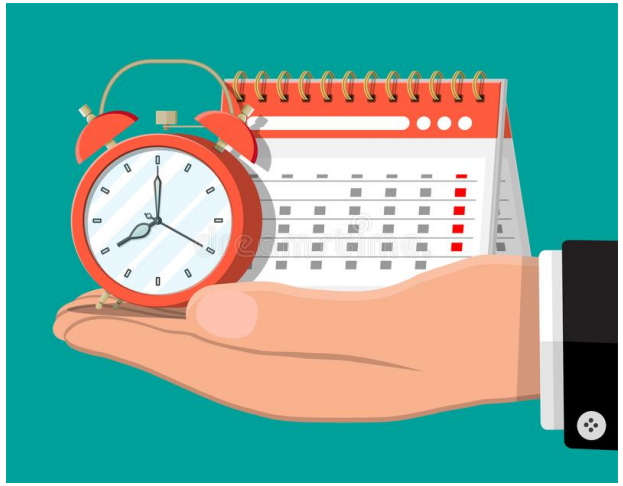
1. Find a quiet place, free of as many distractions as possible. Headphones are recommended.
2. Stay on mute unless you are speaking.
3. Remove or silence alerts from cell phones, e-mail pop-ups, etc.
4. Participate in activities and ask questions. This will be interactive!
5. Give your honest feedback so we can troubleshoot problems and improve the course.

# Getting to know your classmates

- Let's head into a breakout room and introduce ourselves
- You'll have 5-10 minutes to exchange your names and departments and talk about what problems you hope to solve by taking this course
- When you come back, be ready to share 1-2 topic areas that came up as project interests with the whole group!



# How we teach



- We'll walk through the concepts and code together, then you'll have the opportunity to answer questions and practice
- You should have the following:
  - Code files to follow along with the slides
  - Links to interactive knowledge checks
  - Exercise files (we give you 2 files and one has the answers)
- Recordings will be made available

# Module completion checklist

| Objective | Complete |
|---|---|
| Explain the concept of NLP and how it is used in industry | |
| Review the tools and packages in Python 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 | |

# What are NLP and text analysis?

- **Natural language processing (NLP)** is host of methods and disciplines, at the crossroad of :
  - data science
  - computational linguistics
  - artificial intelligence
- NLP's main focus is *processing all forms of human language (spoken or written) and getting insightful information out of it*
- Used for :
  - 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 we can use it for written text

# Why text analysis?

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

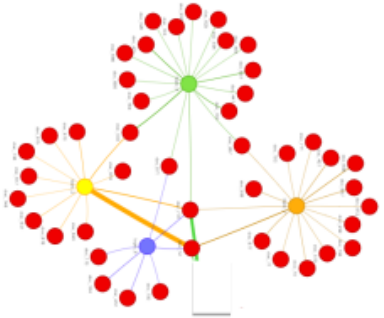
# Text analysis: flow and basic methods

- 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 you perform those steps, you can use more complex methods

Classify documents

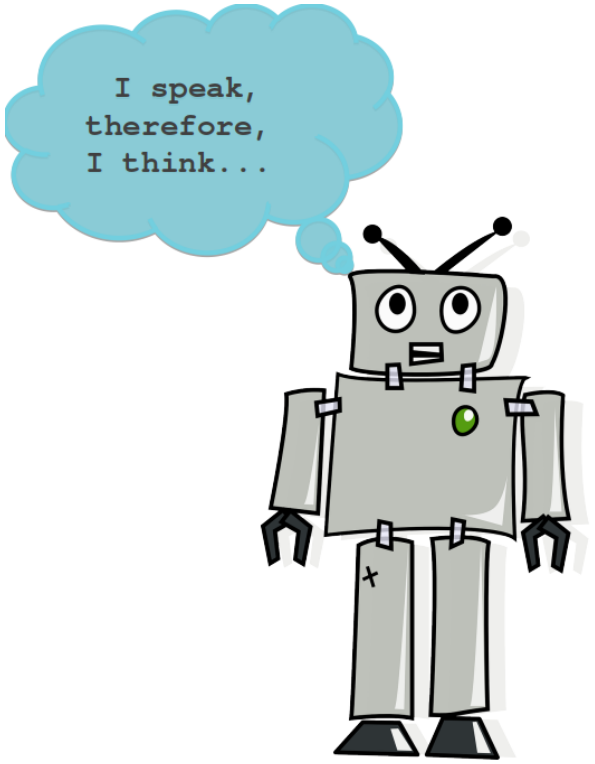


Extract entities

Analyze sentiment

# 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 | ... |



# Use cases in social sciences

## Intro to Text Analysis

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

# Use cases in social sciences

## Intro to Text Analysis

- In the exercises, you will be working with **UN agreement titles**
  - To learn about working with text data using Python
  - To detect patterns in text using 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 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 | |

# Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into `variables`
- Let the `main_dir` be the variable corresponding to your `booz-allen-hamilton` folder

```
from pathlib import Path
# Set `home_dir` to the root directory of your computer.
home_dir = Path.home()

# Set `main_dir` to the location of your `booz-allen-hamilton` folder.
main_dir = home_dir / "Desktop" / "booz-allen-hamilton"

# Make `data_dir` from the `main_dir` and remainder of the path to data directory.
data_dir = main_dir / "data"
```

# Working directory

- Set working directory to the `data_dir` variable we set

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

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

```
/home/[user-name]/Desktop/booz-allen-hamilton/data
```

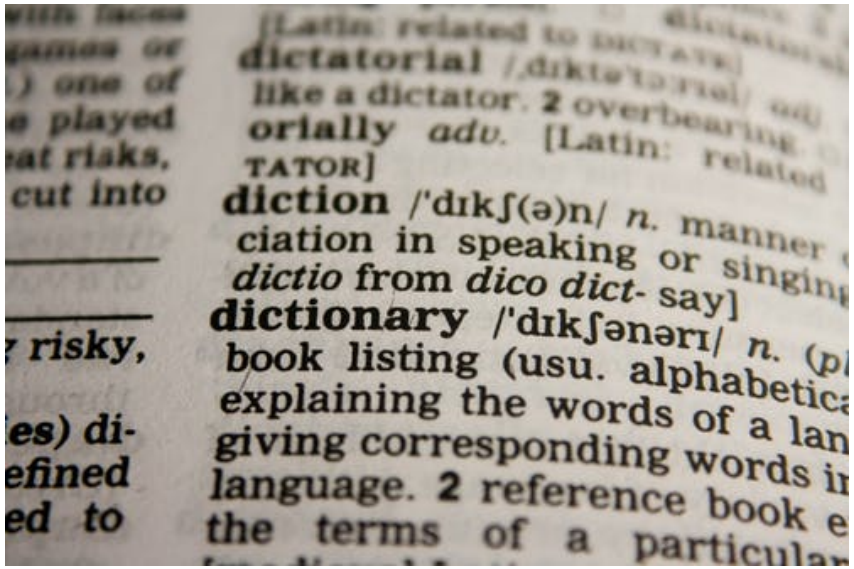
# Loading packages

```
# 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.
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
```

# A little about NLTK

- We will be using NLTK to dive into NLP and text analysis
- Python's **Natural Language Tool Kit**, or **NLTK**, is a very powerful platform for teaching, and working in, **computational linguistics using Python**
- It is free, open source, easy to use, supported by a large community and ***well documented***



# Knowledge check 1

# Module completion checklist

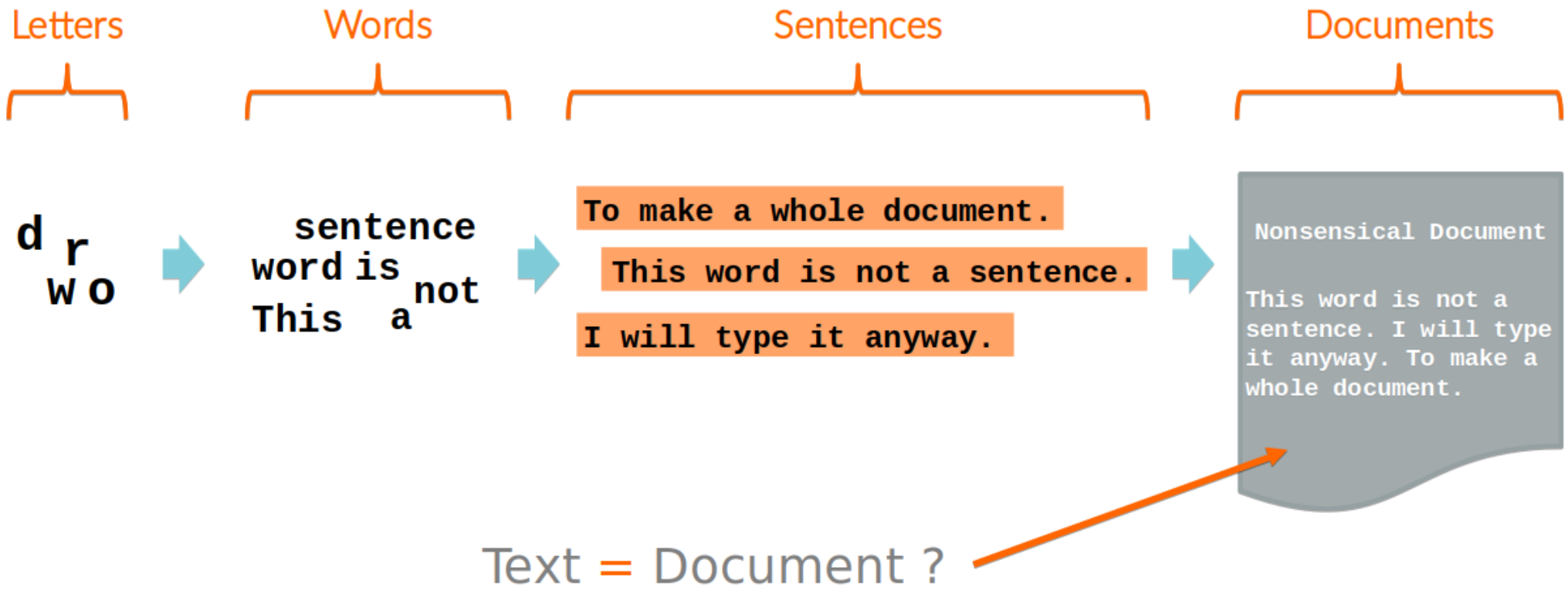
| Objective | Complete |
|---|---|
| Explain the concept of NLP and how it is used in industry | ✔ |
| Review the tools and packages in Python to work with text data | ✔ |
| Discuss what a corpus is based on use cases | |
| Create and inspect a corpus object using NLTK | |
| Load data into Python using 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 | |

# Chat discussion

- Before we begin this section, take a couple of minutes to imagine what your dream text mining dataset might be
- What **collection of text** would you most want to analyze, either for work or just for fun?

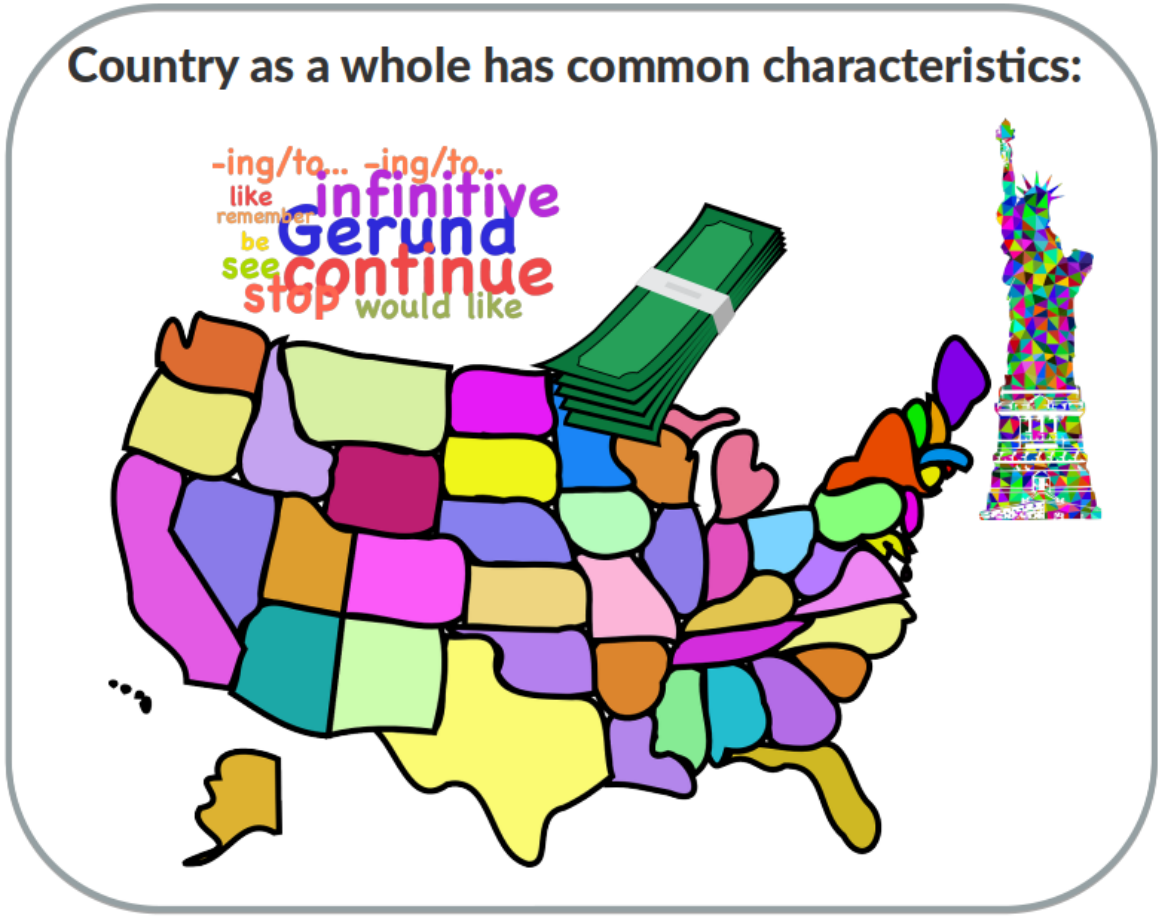
- Type the name of your **dream text mining dataset** (e.g. **"the complete transcripts of *The Real Housewives*"**) into the chat window

# Units of text data

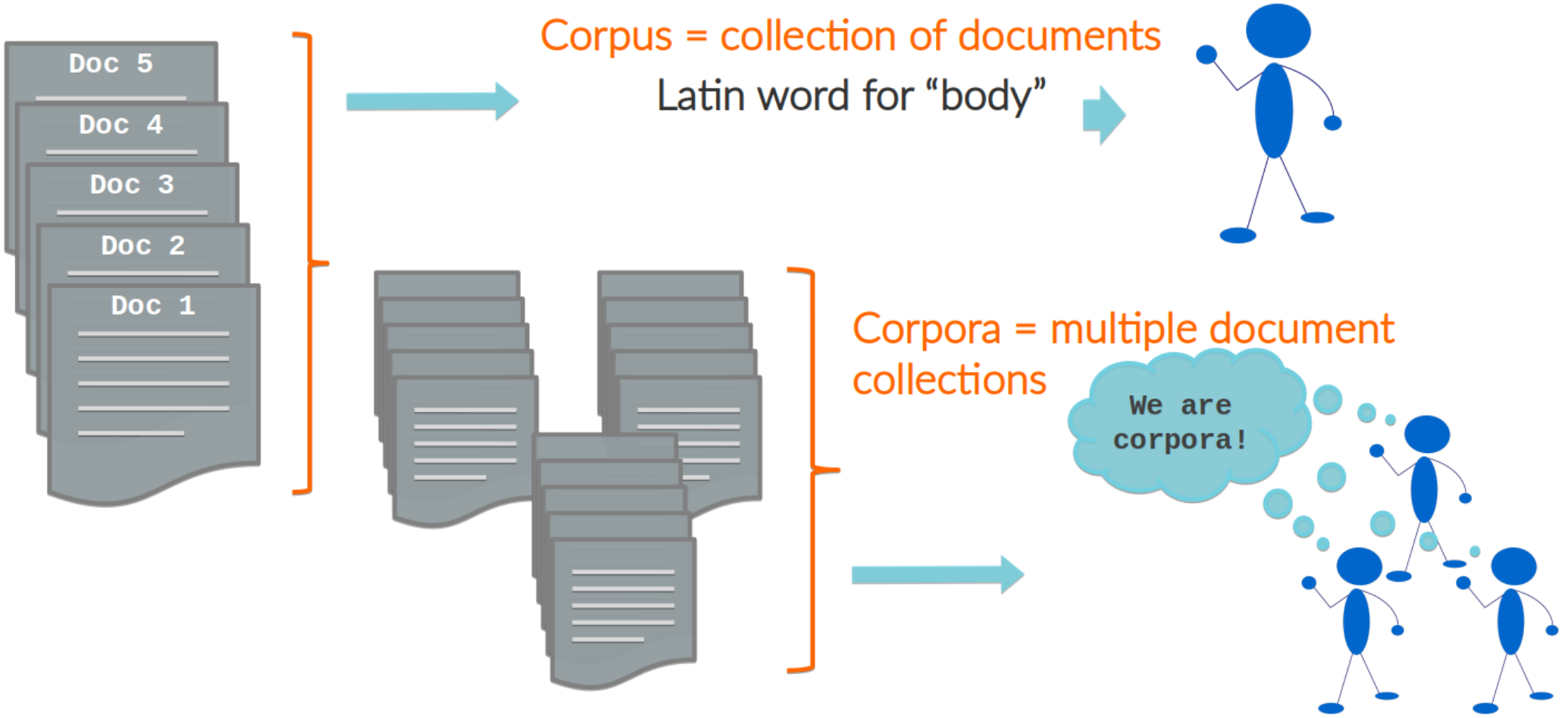


# Units of text data: corpus


≈



# 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

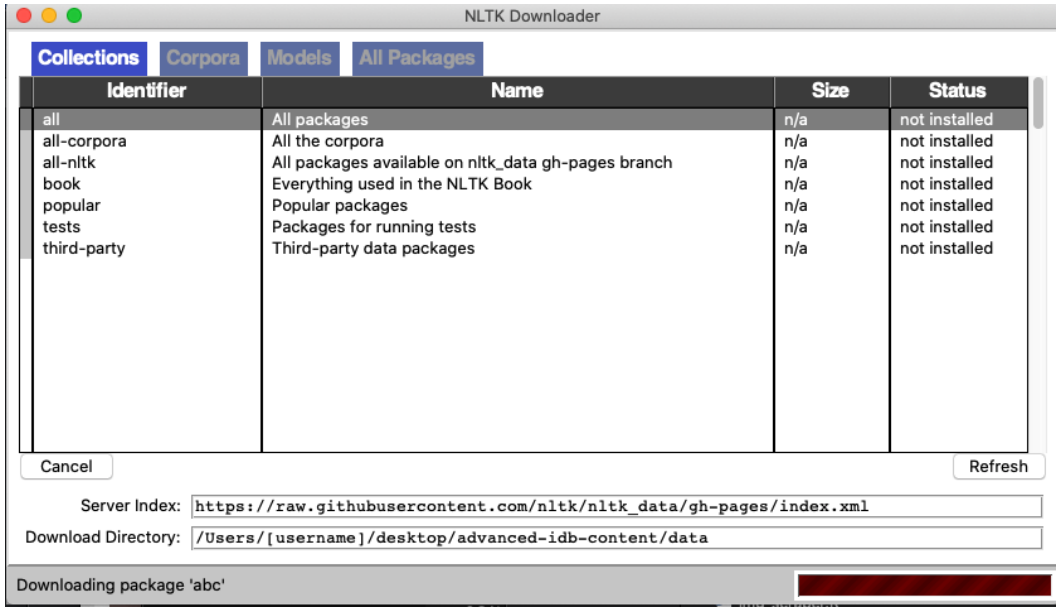
- In this case, we will consider
  - Each **article snippet** as a **document**
  - **All article snippets** together as a **corpus**
- This division of units of text will help us
  - Evaluate **each snippet** for its **subject** and potential **topic**
  - Evaluate **all snippets** for general word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

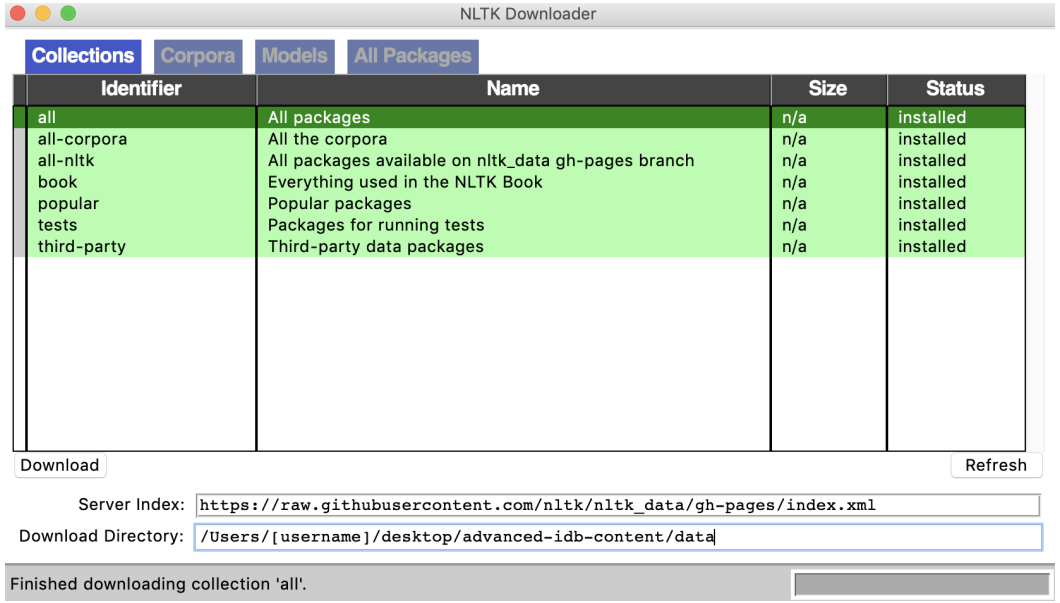
# Module completion checklist

| Objective | Complete |
|---|---|
| Explain the concept of NLP and how it is used in industry | ✔ |
| Review the tools and packages in Python 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 | |

# Download NLTK resources

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




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

# Loading text data

```
# Load corpus from a csv (for Mac).
NYT = pd.read_csv('NYT_article_data.csv')
```

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

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

- For now, we will focus on the `snippet` column
- The other columns will be useful later in our analysis

# The data at first glance

```
# 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 snippet column, 0-15.
print(NYT["snippet"][0:15])
```

```
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...
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 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 | |

# 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. This matrix is the **numeric representation** of your text corpus
6. Document similarity is based on the **similar words** that appear in **documents** with similar **meaning**

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

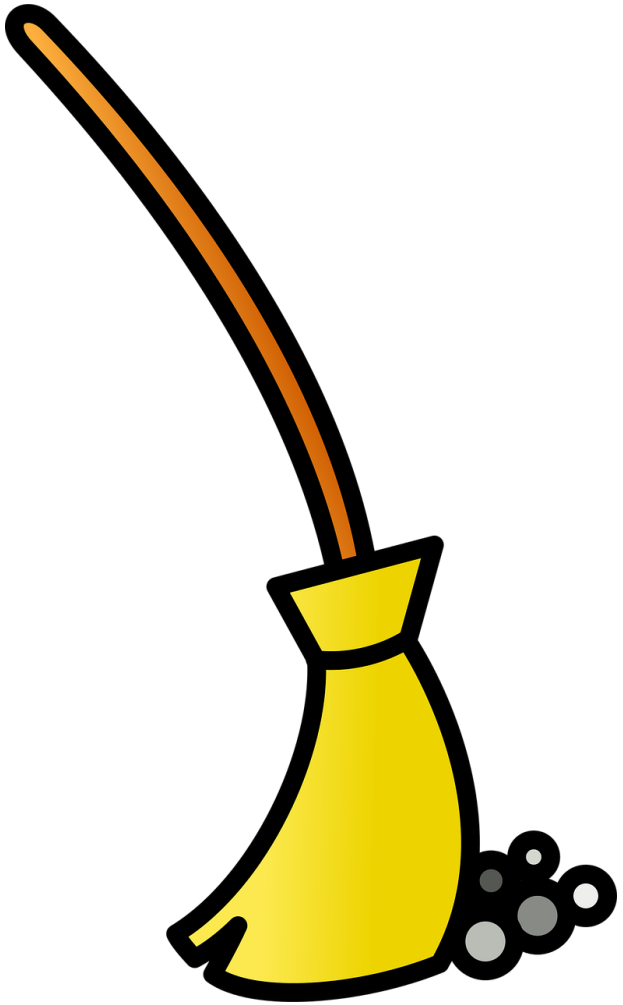
# "Bag-of-words" analysis: key elements

| What we need | What we have learned |
| --- | --- |
| A corpus of documents cleaned and processed in a certain way:<br>• All words are converted to lower case<br>• All punctuation, numbers and special characters are removed<br>• Stopwords are removed<br>• 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 white space (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 all 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 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 | |

# 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

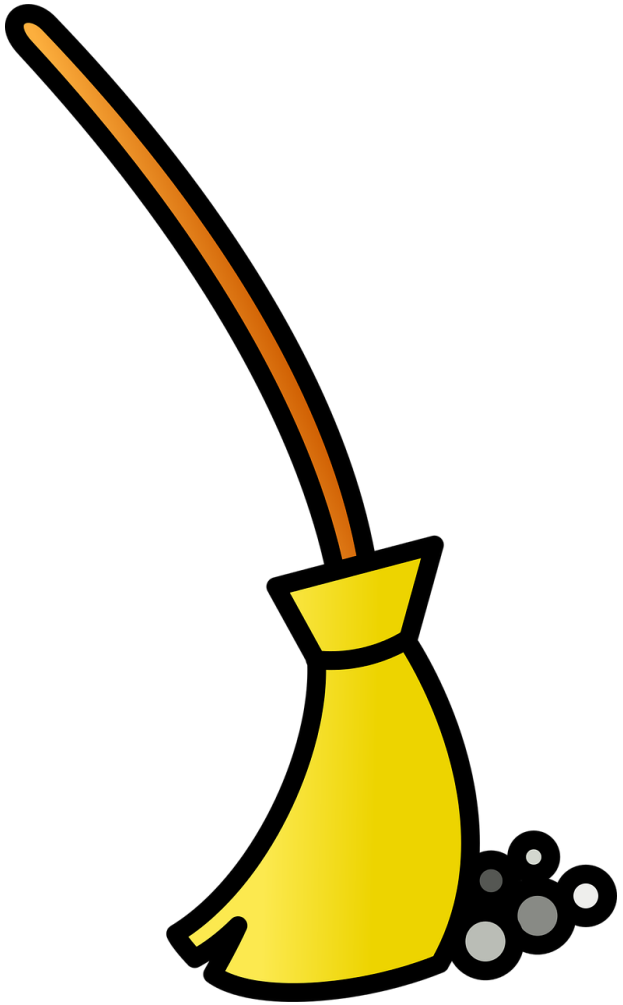
```
# Let's take a look at the first tokenized snippet.
snippet_words = NYT_tokenized[0]
print(snippet_words)
```

```
['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 white space (if needed)



# Convert characters to lower case

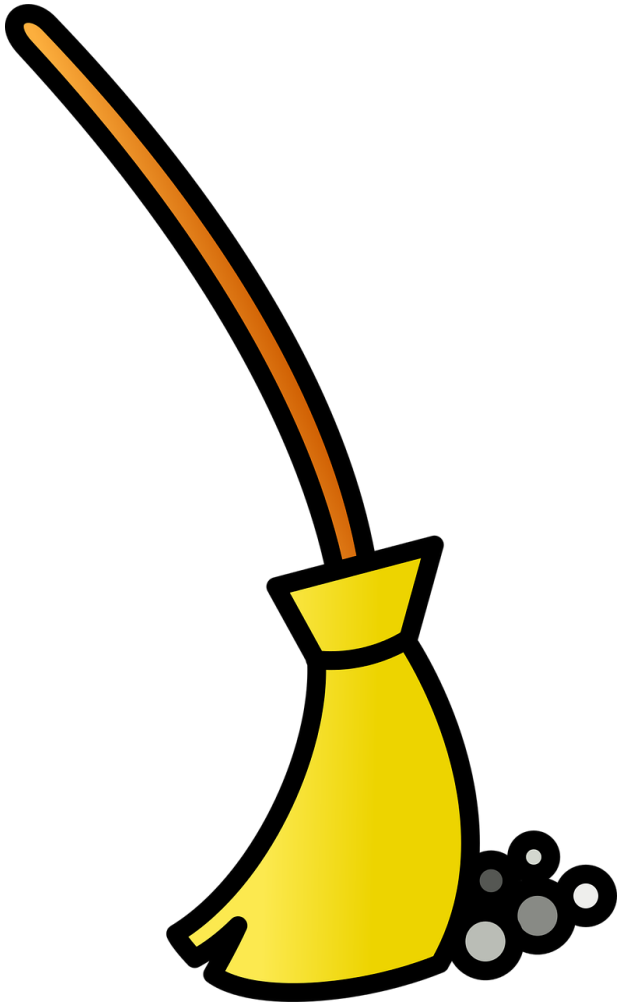
- To convert characters to lower case, we will use `.lower()` function
  - We call the method this way: `character_string.lower()`
- Since every element of the `snippet_words` list is a character string, we will convert each word in the snippet using a **list comprehension**

```
# 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 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 together into coherent sentences
- Such words are the most frequent, and they usually need to be taken out, since 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])
```

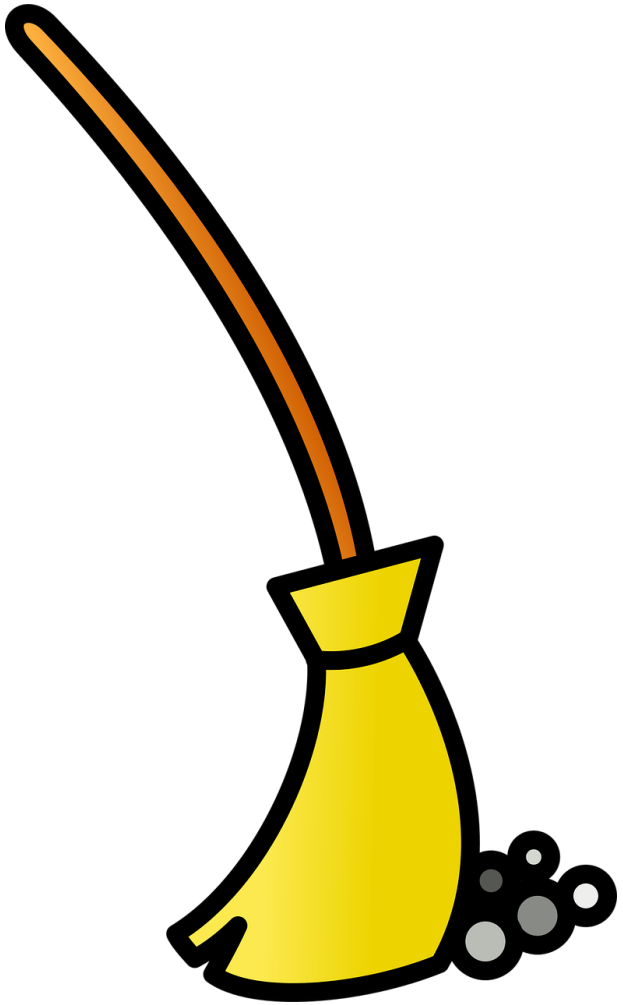
```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]
```

```
# Remove stop words.
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 white space (if needed)



# Remove non-alphabetical characters

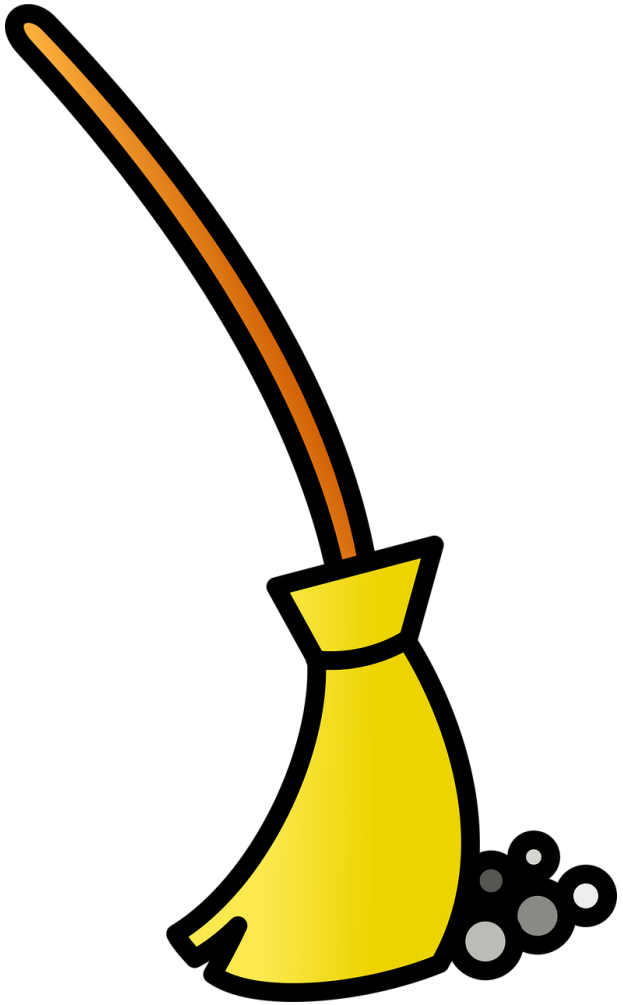
- For text analysis based on "bag-of-words" approach, we only use words
- Numbers, punctuation, and anything other than alphabetical characters must be removed
- We will use `.isalpha()` method to check whether the characters are alphabetical or not
  - We call the method this way: `character_string.isalpha()`
  - 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']
```

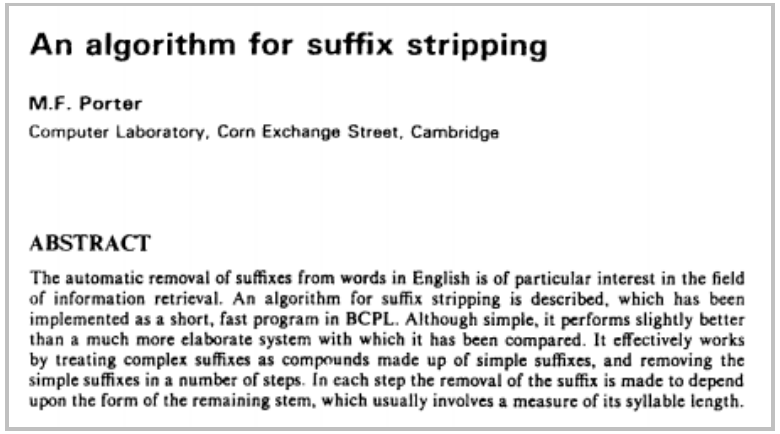
# "Bag-of-words" analysis: cleaning text flow

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 white space (if needed)



# Porter stemmer

- **Stemming** reduces words to their **root** form, allowing 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



**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 cited 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 stop words.
    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])
```

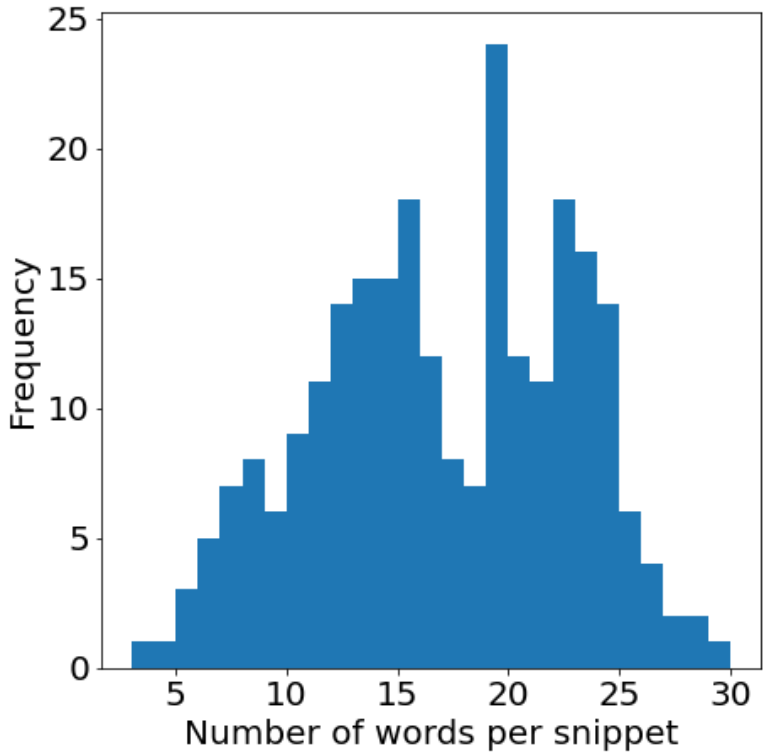
```
['taiwanes', 'presid', 'tsai', 'appoint', 'close', 'polit', 'alli', 'premier', 'cabinet', 'reshuffl']
```

# 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 number of unique values in the list.
plt.hist(word_counts_per_snippet, bins =
len(set(word_counts_per_snippet)))
plt.xlabel('Number of words per snippet')
plt.ylabel('Frequency')
```



# Removing empty and very short snippets (cont'd)

- After cleaning, check 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)
```

```
<string>:1: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a
list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you
meant to do this, you must specify 'dtype=object' when creating the ndarray
```

```
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 will now only keep all of the 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
```

- 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[: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']]
```

# .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 = "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<br>• All words are converted to lower case<br>• All punctuation, numbers, and special characters are removed<br>• Stopwords are removed<br>• 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) | |

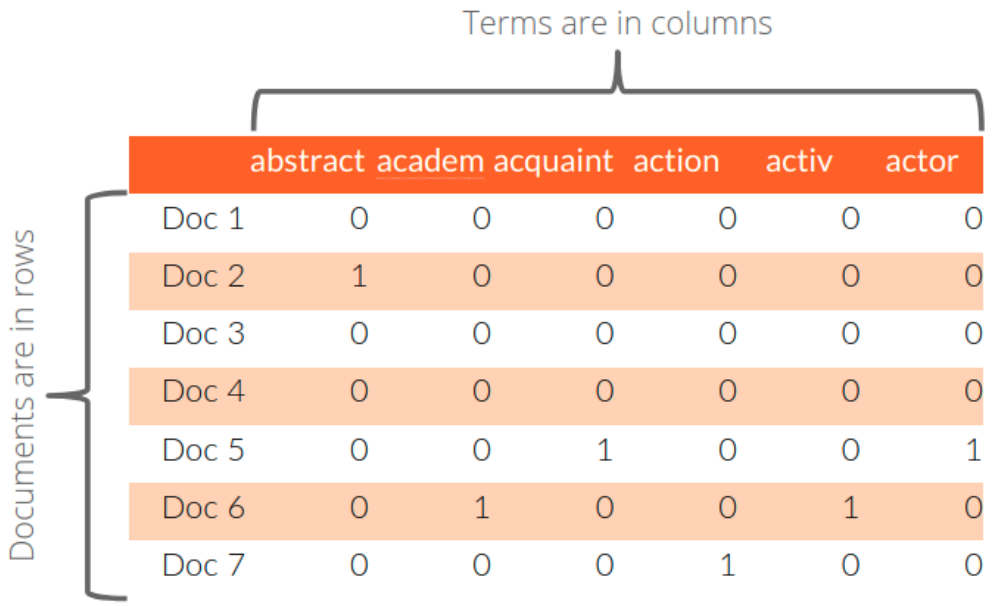
# Knowledge check 3

# 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 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 | |

# 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 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)
- A Term-Document matrix will be just the transpose of the Document-Term matrix, with terms in rows and documents in columns

# 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 used heavily for machine learning
- you can find complete documentation ***here***

# Create DTM with CountVectorizer (cont'd)

- `CountVectorizer` 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***<br>An iterable which yields either str, unicode or file objects. |
| **Returns:** | **X : *array, [n_samples, n_features]***<br>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 can use `lambda` within the 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]))
```

# 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<br>• All words are converted to lower case<br>• All punctuation, numbers, and special characters are removed<br>• Stopwords are removed<br>• 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) | |

# Pickle - what?

- The last thing we will do is save our objects that we will need for the next module
- 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(word_counts_array, open('word_counts_array.sav', 'wb'))
pickle.dump(NYT_clean, open('NYT_clean.sav', 'wb'))
pickle.dump(NYT_clean_list, open('NYT_clean_list.sav', 'wb'))
pickle.dump(corpus_freq_dist, open('corpus_freq_dist.sav', 'wb'))
pickle.dump(X, open('DTM_matrix.sav', 'wb'))
pickle.dump(valid_snippets, open('valid_snippets.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 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 | ✔ |

# Summary

- In this module we discussed the foundations of text mining, identified core terminology used in text mining, and explored the NLTK library as a way to create and inspect a corpus
- We also learned to implement steps to pre-process text data using bag of words analysis method and created term document matrix

- In the next module, we will learn to visualize the distribution of words in a corpus, understand what N-grams are, and weight text data with term frequency inverse document frequency (TF-IDF). Stay excited!

# This completes our module

**Congratulations!**