Application of Deep Learning to Text and Image Data

Module 2, Lab 1: Processing Text

In this notebook, you will learn techniques to analyze and process text data. Text processing is known as *natural language processing (NLP)* and is an important topic because of how much information is communicated through text. Knowing how to handle text will help you build models that perform better and are more useful.

You will learn the following:

- What a word cloud is and how to create one
- How to use stemming and lemmatization
- What part-of-speech tagging is and how it impacts text processing
- How to use named entity recognition to sort data

You will be presented with two kinds of exercises throughout the notebook: activities and challenges.

 **Activity**	 **Challenge**
No coding is needed for an activity. You try to understand a concept, answer questions, or run a code cell.	Challenges are where you can practice your coding skills.

Index

Initial Setup

First let's put everything in place.

In [1]:
```
!pip install -U -q -r requirements.txt
```

```
ERROR: pip's dependency resolver does not currently take into account all the pack
ages that are installed. This behaviour is the source of the following dependency
conflicts.
autovizwidget 0.21.0 requires pandas<2.0.0,>=0.20.1, but you have pandas 2.0.3 whi
ch is incompatible.
hdijupyterutils 0.21.0 requires pandas<2.0.0,>=0.17.1, but you have pandas 2.0.3 w
hich is incompatible.
sparkmagic 0.21.0 requires pandas<2.0.0,>=0.17.1, but you have pandas 2.0.3 which
is incompatible.
```

Install the spaCy library. This will be used to perform some NLP tasks in the lab.

In [2]:
```
!python -m spacy download en_core_web_sm
```

```
/home/ec2-user/anaconda3/envs/pytorch_p310/lib/python3.10/site-packages/torch/cud
a/__init__.py:551: UserWarning: Can't initialize NVML
  warnings.warn("Can't initialize NVML")
Collecting en-core-web-sm==3.7.1
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_
web_sm-3.7.1/en_core_web_sm-3.7.1-py3-none-any.whl (12.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.8/12.8 MB 11.2 MB/s eta 0:00:000
0:010:01
Requirement already satisfied: spacy<3.8.0,>=3.7.2 in /home/ec2-user/anaconda3/env
s/pytorch_p310/lib/python3.10/site-packages (from en-core-web-sm==3.7.1) (3.7.2)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in /home/ec2-user/anaco
nda3/envs/pytorch_p310/lib/python3.10/site-packages (from spacy<3.8.0,>=3.7.2->en-
core-web-sm==3.7.1) (3.0.12)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /home/ec2-user/anaco
nda3/envs/pytorch_p310/lib/python3.10/site-packages (from spacy<3.8.0,>=3.7.2->en-
core-web-sm==3.7.1) (1.0.5)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /home/ec2-user/anacond
a3/envs/pytorch_p310/lib/python3.10/site-packages (from spacy<3.8.0,>=3.7.2->en-co
re-web-sm==3.7.1) (1.0.10)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /home/ec2-user/anaconda3/env
s/pytorch_p310/lib/python3.10/site-packages (from spacy<3.8.0,>=3.7.2->en-core-web
-sm==3.7.1) (2.0.8)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /home/ec2-user/anaconda3/e
nvs/pytorch_p310/lib/python3.10/site-packages (from spacy<3.8.0,>=3.7.2->en-core-w
eb-sm==3.7.1) (3.0.9)
Requirement already satisfied: thinc<8.3.0,>=8.1.8 in /home/ec2-user/anaconda3/env
s/pytorch_p310/lib/python3.10/site-packages (from spacy<3.8.0,>=3.7.2->en-core-web
-sm==3.7.1) (8.2.3)
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in /home/ec2-user/anaconda3/en
vs/pytorch_p310/lib/python3.10/site-packages (from spacy<3.8.0,>=3.7.2->en-core-we
b-sm==3.7.1) (1.1.2)
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in /home/ec2-user/anaconda3/env
s/pytorch_p310/lib/python3.10/site-packages (from spacy<3.8.0,>=3.7.2->en-core-web
-sm==3.7.1) (2.4.8)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /home/ec2-user/anaconda
3/envs/pytorch_p310/lib/python3.10/site-packages (from spacy<3.8.0,>=3.7.2->en-cor
e-web-sm==3.7.1) (2.0.10)
Requirement already satisfied: weasel<0.4.0,>=0.1.0 in /home/ec2-user/anaconda3/en
vs/pytorch_p310/lib/python3.10/site-packages (from spacy<3.8.0,>=3.7.2->en-core-we
b-sm==3.7.1) (0.3.4)
Requirement already satisfied: typer<0.10.0,>=0.3.0 in /home/ec2-user/anaconda3/en
vs/pytorch_p310/lib/python3.10/site-packages (from spacy<3.8.0,>=3.7.2->en-core-we
b-sm==3.7.1) (0.9.4)
Requirement already satisfied: smart-open<7.0.0,>=5.2.1 in /home/ec2-user/anaconda
3/envs/pytorch_p310/lib/python3.10/site-packages (from spacy<3.8.0,>=3.7.2->en-cor
e-web-sm==3.7.1) (6.4.0)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /home/ec2-user/anaconda3/env
s/pytorch_p310/lib/python3.10/site-packages (from spacy<3.8.0,>=3.7.2->en-core-web
-sm==3.7.1) (4.66.1)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /home/ec2-user/anaconda
3/envs/pytorch_p310/lib/python3.10/site-packages (from spacy<3.8.0,>=3.7.2->en-cor
e-web-sm==3.7.1) (2.31.0)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in /home/ec2-u
ser/anaconda3/envs/pytorch_p310/lib/python3.10/site-packages (from spacy<3.8.0,>=
3.7.2->en-core-web-sm==3.7.1) (1.10.13)
Requirement already satisfied: jinja2 in /home/ec2-user/anaconda3/envs/pytorch_p31
```

```
0/lib/python3.10/site-packages (from spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1)
(3.1.2)
Requirement already satisfied: setuptools in /home/ec2-user/anaconda3/envs/pytorch
_p310/lib/python3.10/site-packages (from spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.
1) (68.2.2)
Requirement already satisfied: packaging>=20.0 in /home/ec2-user/anaconda3/envs/py
torch_p310/lib/python3.10/site-packages (from spacy<3.8.0,>=3.7.2->en-core-web-sm=
=3.7.1) (21.3)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /home/ec2-user/anaconda
3/envs/pytorch_p310/lib/python3.10/site-packages (from spacy<3.8.0,>=3.7.2->en-cor
e-web-sm==3.7.1) (3.4.0)
Requirement already satisfied: numpy>=1.19.0 in /home/ec2-user/anaconda3/envs/pyto
rch_p310/lib/python3.10/site-packages (from spacy<3.8.0,>=3.7.2->en-core-web-sm==
3.7.1) (1.23.5)
Requirement already satisfied: language-data>=1.2 in /home/ec2-user/anaconda3/env
s/pytorch_p310/lib/python3.10/site-packages (from langcodes<4.0.0,>=3.2.0->spacy<
3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (1.2.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /home/ec2-user/anaconda
3/envs/pytorch_p310/lib/python3.10/site-packages (from packaging>=20.0->spacy<3.8.
0,>=3.7.2->en-core-web-sm==3.7.1) (3.0.9)
Requirement already satisfied: typing-extensions>=4.2.0 in /home/ec2-user/anaconda
3/envs/pytorch_p310/lib/python3.10/site-packages (from pydantic!=1.8,!=1.8.1,<3.0.
0,>=1.7.4->spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (4.8.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /home/ec2-user/anaconda
3/envs/pytorch_p310/lib/python3.10/site-packages (from requests<3.0.0,>=2.13.0->sp
acy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (3.3.1)
Requirement already satisfied: idna<4,>=2.5 in /home/ec2-user/anaconda3/envs/pytor
ch_p310/lib/python3.10/site-packages (from requests<3.0.0,>=2.13.0->spacy<3.8.0,>=
3.7.2->en-core-web-sm==3.7.1) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /home/ec2-user/anaconda3/env
s/pytorch_p310/lib/python3.10/site-packages (from requests<3.0.0,>=2.13.0->spacy<
3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (1.26.18)
Requirement already satisfied: certifi>=2017.4.17 in /home/ec2-user/anaconda3/env
s/pytorch_p310/lib/python3.10/site-packages (from requests<3.0.0,>=2.13.0->spacy<
3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (2023.7.22)
Requirement already satisfied: blis<0.8.0,>=0.7.8 in /home/ec2-user/anaconda3/env
s/pytorch_p310/lib/python3.10/site-packages (from thinc<8.3.0,>=8.1.8->spacy<3.8.
0,>=3.7.2->en-core-web-sm==3.7.1) (0.7.11)
Requirement already satisfied: confection<1.0.0,>=0.0.1 in /home/ec2-user/anaconda
3/envs/pytorch_p310/lib/python3.10/site-packages (from thinc<8.3.0,>=8.1.8->spacy<
3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (0.1.4)
Requirement already satisfied: click<9.0.0,>=7.1.1 in /home/ec2-user/anaconda3/env
s/pytorch_p310/lib/python3.10/site-packages (from typer<0.10.0,>=0.3.0->spacy<3.8.
0,>=3.7.2->en-core-web-sm==3.7.1) (8.1.7)
Requirement already satisfied: cloudpathlib<0.17.0,>=0.7.0 in /home/ec2-user/anaco
nda3/envs/pytorch_p310/lib/python3.10/site-packages (from weasel<0.4.0,>=0.1.0->sp
acy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (0.16.0)
Requirement already satisfied: MarkupSafe>=2.0 in /home/ec2-user/anaconda3/envs/py
torch_p310/lib/python3.10/site-packages (from jinja2->spacy<3.8.0,>=3.7.2->en-core
-web-sm==3.7.1) (2.1.3)
Requirement already satisfied: marisa-trie>=0.7.7 in /home/ec2-user/anaconda3/env
s/pytorch_p310/lib/python3.10/site-packages (from language-data>=1.2->langcodes<4.
0.0,>=3.2.0->spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (1.1.0)
Installing collected packages: en-core-web-sm
Successfully installed en-core-web-sm-3.7.1
```

```
✓ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
```

In [3]:
```
# Import the dependencies
import nltk
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag
from nltk.stem import SnowballStemmer
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import re, string
import spacy
from spacy import displacy
```

```
Matplotlib is building the font cache; this may take a moment.
/home/ec2-user/anaconda3/envs/pytorch_p310/lib/python3.10/site-packages/torch/cud
a/__init__.py:551: UserWarning: Can't initialize NVML
  warnings.warn("Can't initialize NVML")
```

Next, you need to create a function to preprocess text so that only real words, not special characters and numbers, are displayed.

In [4]:
```
# Preprocess text
def preProcessText(text):
    # Lowercase and strip leading and trailing white space
    text = text.lower().strip()

    # Remove HTML tags
    text = re.compile("<.*?>").sub("", text)

    # Remove punctuation
    text = re.compile("[%s]" % re.escape(string.punctuation)).sub(" ", text)

    # Remove extra white space
    text = re.sub("\s+", " ", text)

    # Remove numbers
    text = re.sub(r"[0-9]", "", text)

    return text
```

Word cloud

Word clouds, which are also known as *text clouds* or *tag clouds*, help you visualize text data by highlighting the important words or phrases. Word clouds convey crucial information at a glance by making commonly occurring words bigger and bolder. These clouds are commonly used to compare and contrast two pieces of text. Word clouds are also used to identify the topic of a document.

To create a word cloud, you will use WordCloud for Python.

The following text is from the What Is Natural Language Processing (NLP)? page on aws.amazon.com.

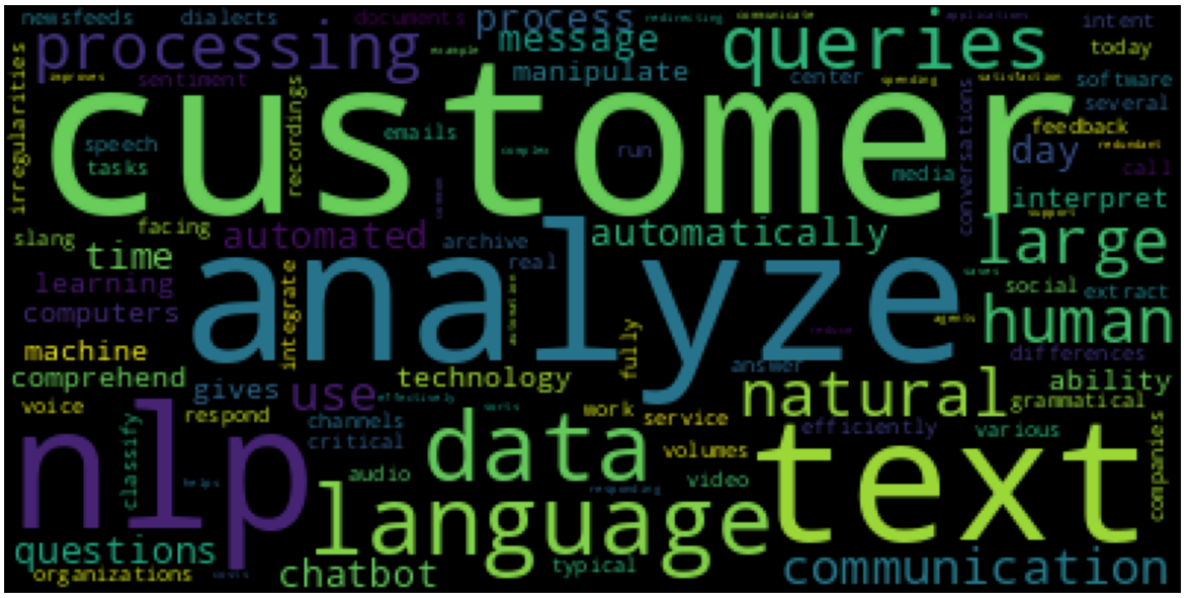
In [5]:
```
text = "Natural language processing (NLP) is a machine learning technology that giv
ability to interpret, manipulate, and comprehend human language. Organizations toda
of voice and text data from various communication channels like emails, text messag
newsfeeds, video, audio, and more. They use NLP software to automatically process t
the intent or sentiment in the message, and respond in real time to human communica
Natural language processing (NLP) is critical to fully and efficiently analyze text
It can work through the differences in dialects, slang, and grammatical irregularit
day-to-day conversations. \
Companies use it for several automated tasks, such as to: \
<li>Process, analyze, and archive large documents</li> \
<li>Analyze customer feedback or call center recordings</li> \
<li>Run chatbots for automated customer service</li> \
<li>Answer who-what-when-where questions</li> \
<li>Classify and extract text</li> \
You can also integrate NLP in customer-facing applications to communicate more effe
customers. For example, a chatbot analyzes and sorts customer queries, responding a
common questions and redirecting complex queries to customer support. This automati
costs, saves agents from spending time on redundant queries, and improves customer
```

In [6]:
```
# Remove stop words before generating the word cloud
wordcloud = WordCloud(stopwords=STOPWORDS, background_color="black", max_words=300)

# Clean up the text to prevent plotting punctuation and duplicate words (for exampl
wordcloud.generate(preProcessText(text))

plt.figure(figsize=(15, 10))
plt.axis("off")
plt.imshow(wordcloud)
```

Out[6]: <matplotlib.image.AxesImage at 0x7fb66130b2e0>



Now that you have created a word cloud, do you see how it can help you quickly identify key words?

Note that the stop words were removed before the graphic was created. This is important so that words that don't impact the meaning of the text aren't overemphasized. Can you think of some examples of stop words?

In this example, you used the precompiled list of stop words that were curated by the WordCloud for Python project. You can print a list of the stop words to make sure that they cover the stop words that you expect.

In [7]:
```
# Show the list of stop words
", ".join(list(STOPWORDS))
```

Out[7]:
```
"few, of, can, he's, here's, just, k, over, myself, has, under, as, through, were
n't, who's, won't, such, i've, ours, r, otherwise, out, itself, can't, be, and, el
se, then, to, isn't, most, these, were, between, could, nor, aren't, against, sh
e'd, their, that, have, haven't, would, both, where's, why, me, like, than, yours,
at, don't, during, didn't, no, his, she's, you're, by, he'd, did, however, ther
e's, also, themselves, before, hasn't, very, having, had, yourself, some, been, ww
w, other, mustn't, http, they, which, they'll, your, while, any, own, about, the
y're, shan't, if, whom, let's, but, is, we, he'll, you, how, being, we'd, this, af
ter, i'd, you've, hers, in, doing, for, too, each, its, she, get, it's, why's, dow
n, so, when's, an, those, i'm, we're, does, wouldn't, how's, because, there, up,
i, off, on, with, into, from, it, yourselves, further, you'd, he, do, cannot, henc
e, i'll, them, ourselves, or, here, couldn't, doesn't, what, where, hadn't, more,
only, com, wasn't, herself, her, him, we'll, what's, they'd, when, himself, our, a
ll, below, the, she'll, are, you'll, a, they've, same, above, ought, shouldn't, sh
ould, therefore, ever, we've, am, my, shall, was, not, once, that's, again, their
s, until, since, who"
```

Part-of-speech tagging

The process of classifying words into their corresponding part of speech based on definition and context is called *part-of-speech tagging*, which is also known as *POS tagging*. A part-of-speech tagger processes a sequence of words and attaches a part-of-speech tag to each word.

For this lab, you will use the nltk.tag package from the Natural Language Toolkit (NLTK). Tagged tokens are encoded as tuples (tag, token). For example, the following tagged token combines the word *fly* with a noun part of speech tag, *NN*: `tagged_tok = ('fly', 'NN')`.

The following table provides the meanings for the tags that the NLTK tagger uses.

Tag	Meaning
CC	Coordinating conjunction

Tag	Meaning
CD	Cardinal digit
DT	Determiner
EX	Existential "there" (examples: “there is,” “there exists”)
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective (example: big)
JJR	Adjective, comparative (example: bigger)
JJS	Adjective, superlative (example: biggest)
LS	List item marker
MD	Modal (examples: could, will)
NN	Noun, singular or mass (example: desk)
NNS	Noun, plural (example: desks)
NNP	Proper noun, singular (example: Harrison)
NNPS	Proper noun, plural (example: Americans)
PDT	Predeterminer (example: "all" the kids)
POS	Possessive ending (example: parent’s)
PRP	Personal pronoun (examples: I, he, she)
RB	Adverb (examples: very, silently)
RBR	Adverb, comparative (example: better)
RBS	Adverb, superlative (example: best)
RP	Particle
TO	"To" (example: "to" the store)
VB	Verb, base form (example: take)
VBD	Verb, past tense (example: took)
VBG	Verb, gerund or present participle (example: taking)
VBN	Verb, past participle (example: taken)
VBP	Verb, singular present, non-third person (example: take)
VBZ	Verb, singular present, third person (example: takes)
WDT	Wh- determiner (example: which)
WP	Wh- pronoun (examples: who, what)
WP$	Possessive wh- pronoun (example: whose)
WRB	Wh- adverb (examples: where, when)

In [8]:
```
# Download resources for the following examples
nltk.download("punkt")
nltk.download("averaged_perceptron_tagger")
nltk.download("maxent_ne_chunker")
nltk.download("words")
nltk.download("wordnet")
```

```
[nltk_data] Downloading package punkt to /home/ec2-user/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /home/ec2-user/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /home/ec2-user/nltk_data...
[nltk_data]   Unzipping chunkers/maxent_ne_chunker.zip.
[nltk_data] Downloading package words to /home/ec2-user/nltk_data...
[nltk_data]   Unzipping corpora/words.zip.
[nltk_data] Downloading package wordnet to /home/ec2-user/nltk_data...
```

Out[8]:
```
True
```

Now you can use the tagger to tag each token or word in the following text.

Important: Always remember to preprocess the text before tagging, as we have done before in this notebook.

In [ ]:
```
# Text sample
text
```

Try it yourself!

To use the NLTK part-of-speech tagger, run the following cell.

Observe the tags that are assigned to each word, and use the table from a previous cell to understand the meaning of each tag.

In [9]:
```
# Part-of-speech tagging
pos_tag(word_tokenize(preProcessText(text)))
```

Out[9]:
```
[('natural', 'JJ'),
 ('language', 'NN'),
 ('processing', 'NN'),
 ('nlp', 'NN'),
 ('is', 'VBZ'),
 ('a', 'DT'),
 ('machine', 'NN'),
 ('learning', 'VBG'),
 ('technology', 'NN'),
 ('that', 'WDT'),
 ('gives', 'VBZ'),
 ('computers', 'NNS'),
 ('the', 'DT'),
 ('ability', 'NN'),
 ('to', 'TO'),
 ('interpret', 'VB'),
 ('manipulate', 'NN'),
 ('and', 'CC'),
 ('comprehend', 'VB'),
 ('human', 'JJ'),
 ('language', 'NN'),
 ('organizations', 'NNS'),
 ('today', 'NN'),
 ('have', 'VBP'),
 ('large', 'JJ'),
 ('volumes', 'NNS'),
 ('of', 'IN'),
 ('voice', 'NN'),
 ('and', 'CC'),
 ('text', 'NN'),
 ('data', 'NNS'),
 ('from', 'IN'),
 ('various', 'JJ'),
 ('communication', 'NN'),
 ('channels', 'NNS'),
 ('like', 'IN'),
 ('emails', 'NNS'),
 ('text', 'JJ'),
 ('messages', 'NNS'),
 ('social', 'JJ'),
 ('media', 'NNS'),
 ('newsfeeds', 'NNS'),
 ('video', 'VBP'),
 ('audio', 'JJ'),
 ('and', 'CC'),
 ('more', 'RBR'),
 ('they', 'PRP'),
 ('use', 'VBP'),
 ('nlp', 'JJ'),
 ('software', 'NN'),
 ('to', 'TO'),
 ('automatically', 'RB'),
 ('process', 'VB'),
 ('this', 'DT'),
 ('data', 'NN'),
 ('analyze', 'VBZ'),
```

```
('the', 'DT'),
('intent', 'NN'),
('or', 'CC'),
('sentiment', 'NN'),
('in', 'IN'),
('the', 'DT'),
('message', 'NN'),
('and', 'CC'),
('respond', 'NN'),
('in', 'IN'),
('real', 'JJ'),
('time', 'NN'),
('to', 'TO'),
('human', 'JJ'),
('communication', 'NN'),
('natural', 'JJ'),
('language', 'NN'),
('processing', 'NN'),
('nlp', 'NN'),
('is', 'VBZ'),
('critical', 'JJ'),
('to', 'TO'),
('fully', 'RB'),
('and', 'CC'),
('efficiently', 'RB'),
('analyze', 'JJ'),
('text', 'NN'),
('and', 'CC'),
('speech', 'NN'),
('data', 'NNS'),
('it', 'PRP'),
('can', 'MD'),
('work', 'VB'),
('through', 'IN'),
('the', 'DT'),
('differences', 'NNS'),
('in', 'IN'),
('dialects', 'NNS'),
('slang', 'JJ'),
('and', 'CC'),
('grammatical', 'JJ'),
('irregularities', 'NNS'),
('typical', 'JJ'),
('in', 'IN'),
('day', 'NN'),
('to', 'TO'),
('day', 'NN'),
('conversations', 'NNS'),
('companies', 'NNS'),
('use', 'VBP'),
('it', 'PRP'),
('for', 'IN'),
('several', 'JJ'),
('automated', 'JJ'),
('tasks', 'NNS'),
('such', 'JJ'),
```

```
('as', 'IN'),
('to', 'TO'),
('process', 'VB'),
('analyze', 'NN'),
('and', 'CC'),
('archive', 'JJ'),
('large', 'JJ'),
('documents', 'NNS'),
('analyze', 'VBP'),
('customer', 'NN'),
('feedback', 'NN'),
('or', 'CC'),
('call', 'VB'),
('center', 'NN'),
('recordings', 'NNS'),
('run', 'VBP'),
('chatbots', 'NNS'),
('for', 'IN'),
('automated', 'JJ'),
('customer', 'NN'),
('service', 'NN'),
('answer', 'NN'),
('who', 'WP'),
('what', 'WP'),
('when', 'WRB'),
('where', 'WRB'),
('questions', 'NNS'),
('classify', 'VBP'),
('and', 'CC'),
('extract', 'VBP'),
('text', 'IN'),
('you', 'PRP'),
('can', 'MD'),
('also', 'RB'),
('integrate', 'VB'),
('nlp', 'NN'),
('in', 'IN'),
('customer', 'NN'),
('facing', 'NN'),
('applications', 'NNS'),
('to', 'TO'),
('communicate', 'VB'),
('more', 'RBR'),
('effectively', 'RB'),
('with', 'IN'),
('customers', 'NNS'),
('for', 'IN'),
('example', 'NN'),
('a', 'DT'),
('chatbot', 'NN'),
('analyzes', 'NN'),
('and', 'CC'),
('sorts', 'NNS'),
('customer', 'NN'),
('queries', 'NNS'),
('responding', 'VBG'),
```

```
('automatically', 'RB'),
('to', 'TO'),
('common', 'JJ'),
('questions', 'NNS'),
('and', 'CC'),
('redirecting', 'VBG'),
('complex', 'JJ'),
('queries', 'NNS'),
('to', 'TO'),
('customer', 'NN'),
('support', 'NN'),
('this', 'DT'),
('automation', 'NN'),
('helps', 'VBZ'),
('reduce', 'VB'),
('costs', 'NNS'),
('saves', 'VBZ'),
('agents', 'NNS'),
('from', 'IN'),
('spending', 'NN'),
('time', 'NN'),
('on', 'IN'),
('redundant', 'JJ'),
('queries', 'NNS'),
('and', 'CC'),
('improves', 'VBZ'),
('customer', 'NN'),
('satisfaction', 'NN')]
```

Refer to the table in a previous cell to identify the tags that the NLTK tagger produces.

Stemming and lemmatization

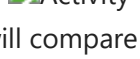
Stemming and lemmatization are two ways to process words so that a model will be more efficient. Both methods remove parts of words so that they can be grouped together.

For example, in the following sentence, "ning" would be removed from "running" so that "running" and "run" would be categorized the same.

The child enjoys **running**, so they **run** every day.

What could make stemming and lemmatization difficult to do properly?

Try it yourself!



In the next few sections, you will compare stemming and lemmatization.
Consider which text processing method is more suitable for the use case that is provided.

Stemming

Stemming is a rule-based system to convert words into their root forms by removing suffixes. This method helps to enhance similarities (if any) between sentences.

Examples:

"jumping", "jumped" -> "jump"

"cars" -> "car"

In [10]:
```
original_text = "   This is a message to be cleaned. It may involve some things lik
print(original_text)
```

```
   This is a message to be cleaned. It may involve some things like: <br>, ?, :,
''  adjacent spaces and tabs     .
```

In [11]:
```
# Cleaned text
cleaned_text = preProcessText(original_text)
print(cleaned_text)
```

```
this is a message to be cleaned it may involve some things like adjacent spaces an
d tabs
```

In [12]:
```
# Use a tokenizer and stemmer from the NLTK library
# Initialize the stemmer
snow = SnowballStemmer("english")

stemmed_sentence = []
# Tokenize the sentence
words = word_tokenize(cleaned_text)
for w in words:
    # Stem the word/token
    stemmed_sentence.append(snow.stem(w))
stemmed_text = " ".join(stemmed_sentence)
```

In [13]:
```
print(stemmed_text)
```

```
this is a messag to be clean it may involv some thing like adjac space and tab
```

From the output of the previous code cell, you can see that stemming isn't perfect. It makes mistakes, such as "messag", "involv", and "adjac". Stemming is a rule-based method that sometimes mistakenly removes suffixes from words. It does run quickly, which makes it appealing to use for massive datasets.

Lemmatization

If you aren't satisfied with the result of stemming, you can use the lemmatization instead. This method usually requires more work but gives better results.

Lemmatization needs to know the correct word position tags, such as "noun", "verb", or "adjective". You need to use another NLTK function to feed this information to the lemmatizer.

The cell below uses part of the full list of position tags listed in the previous session `Part-of-speech tagging`.

In [14]:
```
# Initialize the lemmatizer
wl = WordNetLemmatizer()

# Helper function to map NTLK position tags
# Full list is available here: https://www.ling.upenn.edu/courses/Fall_2003/ling001
def get_wordnet_pos(tag):
    if tag.startswith("J"):
        return wordnet.ADJ
    elif tag.startswith("V"):
        return wordnet.VERB
    elif tag.startswith("N"):
        return wordnet.NOUN
    elif tag.startswith("R"):
        return wordnet.ADV
    else:
        return wordnet.NOUN


lemmatized_sentence = []
# Tokenize the sentence
words = word_tokenize(cleaned_text)
# Get position tags
word_pos_tags = nltk.pos_tag(words)
# Map the position tag and lemmatize the word/token
for idx, tag in enumerate(word_pos_tags):
    lemmatized_sentence.append(wl.lemmatize(tag[0], get_wordnet_pos(tag[1])))

lemmatized_text = " ".join(lemmatized_sentence)
```

In [15]:
```
print(lemmatized_text)
```

```
this be a message to be clean it may involve some thing like adjacent space and ta
b
```

How do the results compare? Is the lemmatized text better than the stemmed text?

Named entity recognition

Named entity recognition involves identification of key information in text and then classifying that information into predefined categories, such as person, organization, place, or date. This is one of the most popular NLP tasks.

For this section, you will use spaCy. The following table lists the categories and meanings of the category labels that the spaCy module uses.

Category	Meaning
CARDINAL	Numerals that don't fall under another type
DATE	Absolute or relative dates or periods
EVENT	Named hurricanes, battles, wars, sports events, and so on
FAC	Buildings, airports, highways, bridges, and so on
GPE	Countries, cities, states
LANGUAGE	Any named language
LAW	Named documents made into laws
LOC	Non-GPE locations, mountain ranges, bodies of water
MONEY	Monetary values, including unit
NORP	Nationalities, or religious or political groups
ORDINAL	"first", "second", and so on
ORG	Companies, agencies, institutions, and so on
PERCENT	Percentage, including "%"
PERSON	People, including fictional
PRODUCT	Objects, vehicles, foods, and so on (not services)
QUANTITY	Measurements, as of weight or distance
WORK_OF_ART	Titles of books, songs, and so on

The following text was retrieved from the Amazon - The Climate Pledge page of the About Amazon website.

In [16]:
```
# Sample text for named entity recognition
ner_text = "Amazon and Global Optimism co-founded The Climate Pledge, \
a commitment to net-zero \
carbon by 2040."
```

In [17]:
```
# Load the spaCy English pipeline for named entity recognition
NER = spacy.load("en_core_web_sm")

# Tag entities in the text
for word in NER(ner_text).ents:
    print(word.text, word.label_)
```

```
Amazon ORG
Global Optimism ORG
The Climate Pledge WORK_OF_ART
2040 DATE
```

Visualizing the Tags

spaCy has a visualizer called displaCy that you can use to visualize tags. Run the following cell to see it working.

In [18]:
```
# Visual tag with text
displacy.render(NER(ner_text), style="ent", jupyter=True)
```

Amazon ORG and Global Optimism ORG co-founded The Climate Pledge WORK_OF_ART , a commitment to net-zero carbon by 2040 DATE .

As you can see, named entity recognition can help you identify different entities in text. The process isn't always correct, but it can process large sections of text faster than a human can.

Conclusion

In this lab, you practiced using text processing techniques.

Next lab

In the next lab, you will learn about the bag-of-words (BoW) method to convert text data into numerical values.