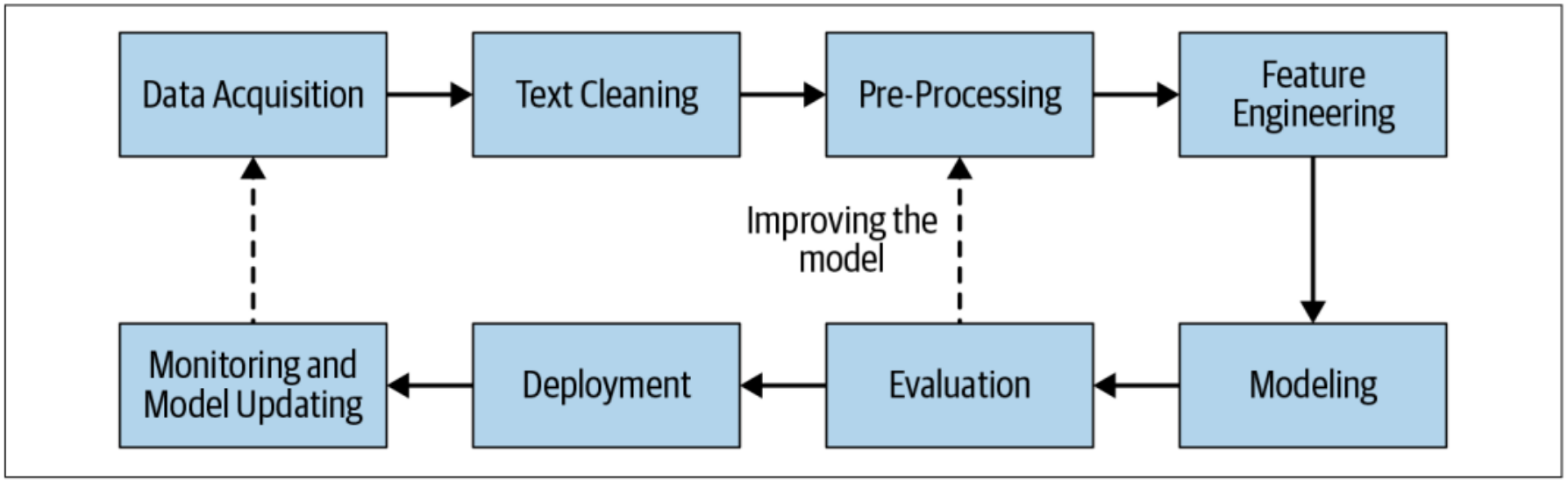
Natural Language Processing (NLP):

Text extraction, cleanup and pre-processing

Text extraction and cleanup

NLP pipeline



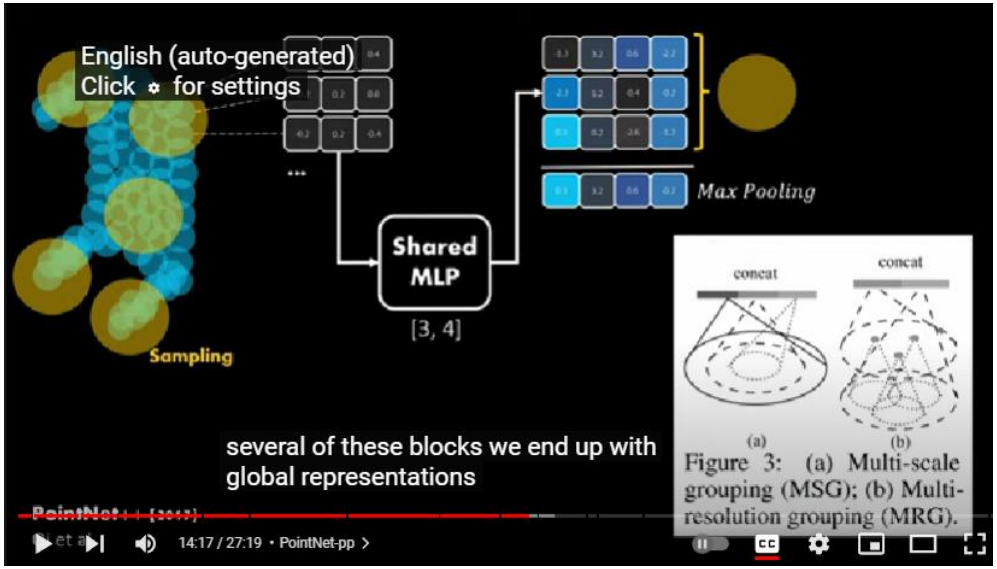

```
<ad-container data-ad-active="true" id="ad-container-1" data-ad-placement-type="mutation_flex" class="ad-container-used ad-empty" data-ad-handled="true" data-ad-active-format="mutation_flex" data-ad-assigned="true" data-ad-unit-id="4469811" data-ad-server="adition" data-rtx-ad-unit-id data-rtx-ad-server data-ad-sizes>…</ad-container>
<main id="main">
<div class="layout">grid
<div class="story js-story">
<ad-container data-ad-active="true" id="ad-container-2" data-ad-placement-type="billboard" class="ad-container-used" data-ad-handled="true" data-ad-active-format="billboard" data-ad-assigned="true" data-ad-unit-id="prebid/728x90" data-ad-server="prebid" data-rtx-ad-unit-id data-rtx-ad-server data-ad-sizes="[[728,90]]">…</ad-container>flex
<article class="story-article">
<div>
<div class="article-header">…</div>
<div class="article-body">
<div data-section-type="gallery" data-position="auto" data-size="auto" data-gallery-type="secondary">…</div>
<p> == $0
"Wien – Die Inflationsentwicklung hilft in der diesjährigen Metaller-Herbstlohnrunde eher den Arbeitgebern. Anders als in früheren Jahren erwarten die Wirtschaftsforscher keine akuten Ausschläge der Teuerungsrate, sondern in den kommenden drei bis vier Monaten ein Verharren auf sehr hohem Niveau. So schwierig es sein mag, die hohe, auf Gas- und Ölpreissteigerungen basierende importierte Inflation in den Erzeugerpreisen unterzubringen – den Unternehmern kommt dies nun entgegen."
</p>
<div class="native-ad">…</div>flex
<p>…</p>
<p>…</p>
<ad-container data-ad-active="true" id="ad-container-5" data-ad-placement-type="inread" class="ad-container-used" data-ad-handled="true" data-ad-active-format="inread" data-ad-assigned="true" data-ad-unit-id="4687935" data-ad-server="adition" data-rtx-ad-unit-id data-rtx-ad-server data-ad-sizes>…</ad-container>
<h3>…</h3>
```

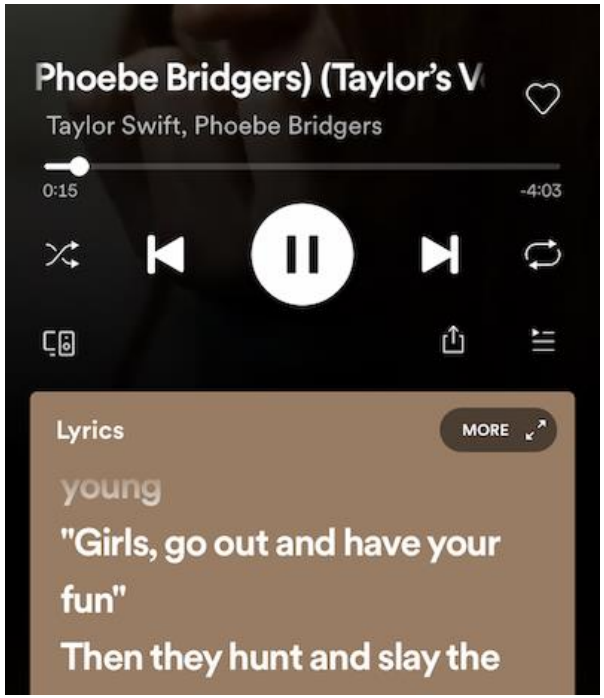

[パク・ヘリ]
[写真＝文化体育観光部]

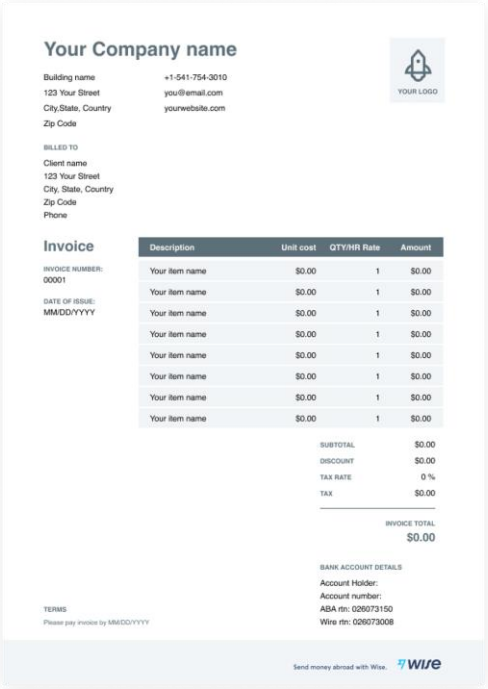
文化体育観光部の朴普均 (パク・ポギュン)長官は１６日、「Culture Bridge Festa」行事が行われたソウル・汝矣島（ヨイド）でベトナムのグエン・バン・フン文化スポーツ観光大臣と会い、文化・体育・観光分野の交流・協力を強化することで一致した。

朴長官は「韓国とベトナムが長い歴史と文化を通じて、相互理解を深めるのであれば、新たな協力の機会を作ることができるだろう」とし、「両国の文化交流や協力イベントに積極的に参加する」と述べた。

また、最近、韓国政府が積極的に進めているアジア・サッカー連盟（ＡＦＣ）の２０２３年アジア・カップの韓国誘致に対するベトナムの支持を要請した。

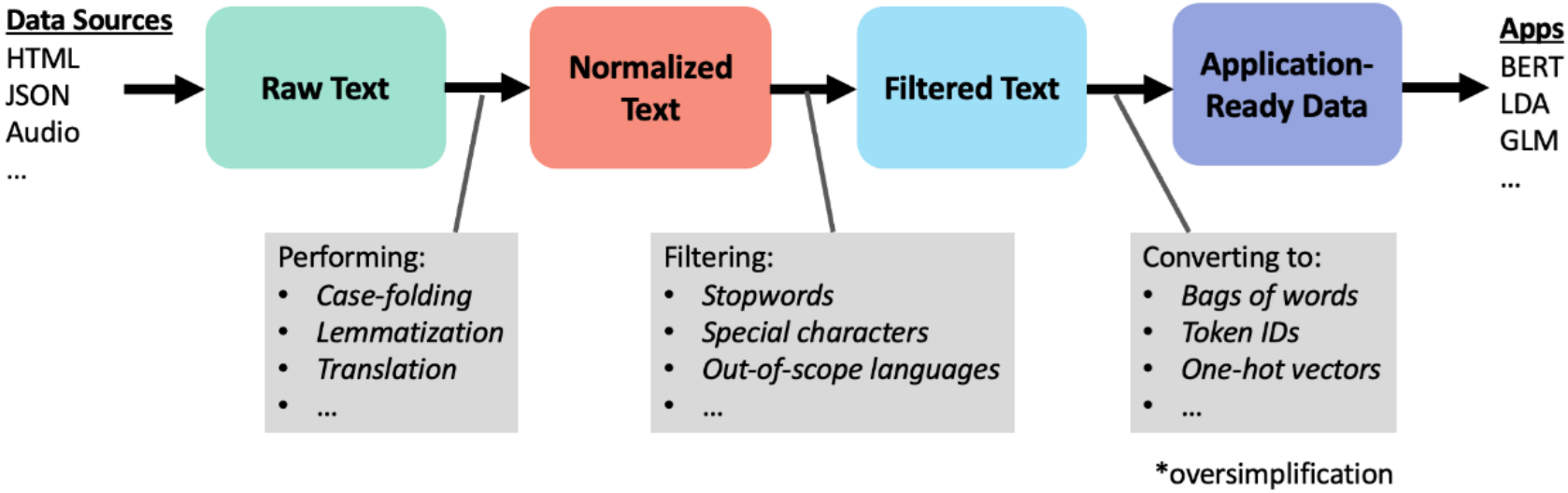



Phoebe Bridgers) (Taylor's V
Taylor Swift, Phoebe Bridgers
0:15
-4:03
Lyrics
MORE
young
"Girls, go out and have your fun"
Then they hunt and slay the


Your Company name
Building name
123 Your Street
City,State, Country
Zip Code
+1-541-754-3010
you@email.com
yourwebsite.com
YOUR LOGO
BILLED TO
Client name
123 Your Street
City, State, Country
Zip Code
Phone
Invoice
INVOICE NUMBER:
00001
DATE OF ISSUE:
MM/DD/YYYY
Description
Unit cost
QTY/HR Rate
Amount
Your item name
$0.00
1
SUBTOTAL
DISCOUNT
TAX RATE
TAX
INVOICE TOTAL
BANK ACCOUNT DETAILS
Account Holder:
Account number:
ABA rtn: 026073150
Wire rtn: 026073008
TERMS
Send money abroad with Wise.
wise


6. Put quality first, cost second. Let our search for better materials be perpetual
7. FOOD FOR INGREDIENTS— NOT ISOLATED NUTRIENTS—UNLESS NOT AVAILABLE NATURALLY
8. LOOK FOR WAYS TO IMPROVE PROCESSES
9. Education before marketing
10. BE THE BEST

Basic NLP pre-processing pipeline*



Text extraction and cleanup

- **Process of extracting raw text from the input data by removing all the other non-textual information (markup, metadata, etc.), and converting the text to the required encoding format**

- Typically, this **depends** on the **format** of available data:
  - Static data from PDF, HTML, or text
  - Some form of continuous data stream
  - Photos, videos, audio files, etc.

- **Text extraction** is an **important** and standard data-wrangling step usually not requiring any NLP-specific techniques
  - It can also be the most time-consuming part of a project

HTML parsing and cleanup

- Say we're working on a project where we want to get economy news from Der Standard, **how can we get the article title and text?**

- First we **fetch the HTML content** of the page (e.g., using the requests library)

- Then, we **parse the HTML document** and **extract relevant data**:
  - By implementing our own parser (usually a much worse option)
  - Using existing libraries (e.g., **Beautiful Soup**, **Scrapy**)
  - Here, we rely on our knowledge of the structure of an HTML document [1]
  - Always **visually inspect** the received HTML document first!

[1] https://www.w3schools.com/html/html_intro.asp

HTML parsing and cleanup

```
from bs4 import BeautifulSoup
from urllib.request import urlopen
myurl = "https://www.derstandard.at/story/2000140021879/lohnverhandlungen-im-\
zeichen-hoher-inflation-metallergewerkschafter-erhoehen-druck"
html = urlopen(myurl).read()
soupified = BeautifulSoup(html, "html.parser")
article_title = soupified.find("h1", {"class": "article-title"})
article_text = soupified.find("div", {"class":"article-text"})
print("Article title: \n", article_title.get_text().strip())
print("Article text: \n", article_text.get_text().strip())
```

```
Article title:
 Lohnverhandlungen im Zeichen hoher Inflation: Metallergewerkschafter erhöhen Dru
Article text:
 Eine Zahl liegt auf dem Tisch – und den Arbeitgebern wohl schwer im Magen. 10,6
```

Character encoding [2]

- **Process of assigning numbers to graphical characters, especially the written characters of language, allowing them to be stored, transmitted, and transformed using digital computers**
  - Morse code, Braille, ASCII, Unicode, etc.

- **Low cost of digital representation of data** in modern computer systems allows more elaborate character codes (e.g., **Unicode**)

- Character encoding using **internationally accepted standards** permits **worldwide interchange of text** in electronic form

[2] https://en.wikipedia.org/wiki/Character_encoding

Character encoding [2]

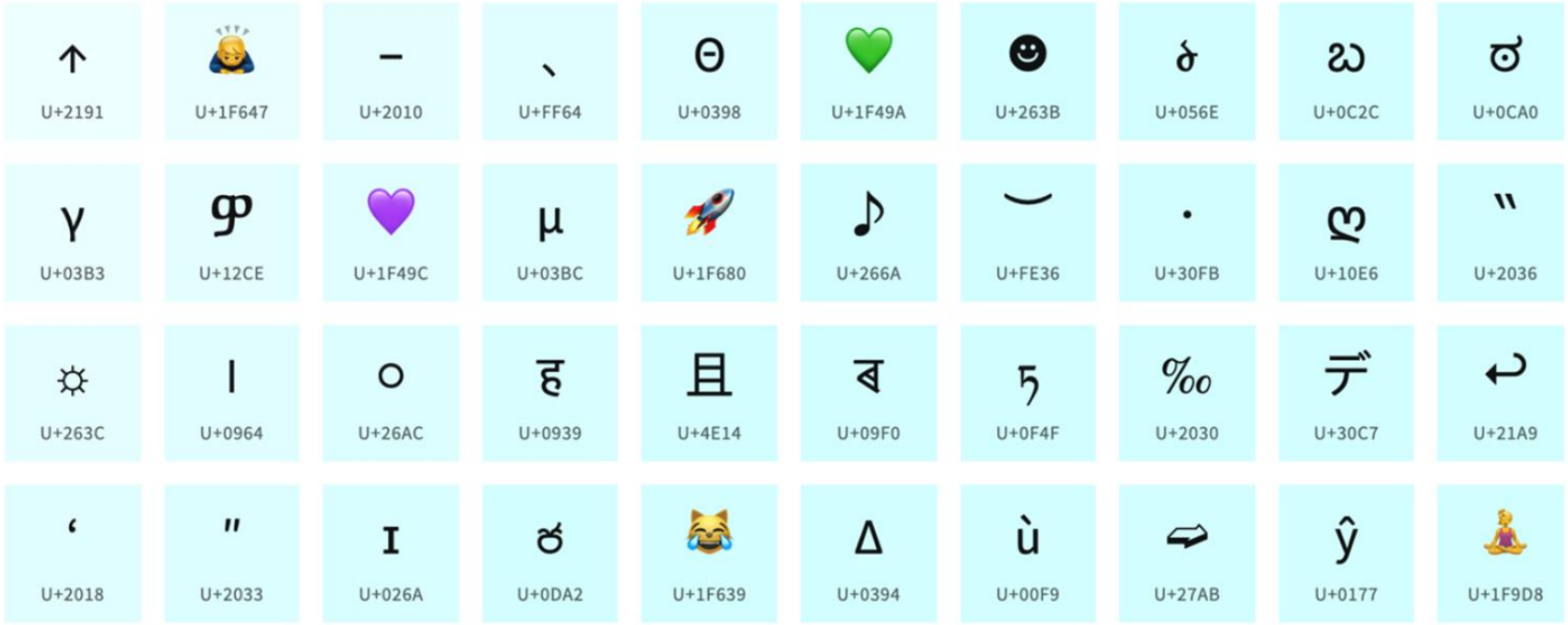
- Most common standards used today:
  - **Unicode**, **GB 18030**, **ISO** encodings, **MS-Windows** encodings

Character	Unicode code point	Glyph
Latin A	U+0041	A
Latin sharp S	U+00DF	ß
Han for East	U+6771	東
Ampersand	U+0026	&
Inverted exclamation mark	U+00A1	¡
Section sign	U+00A7	§

[2] https://en.wikipedia.org/wiki/Character_encoding

Unicode normalization

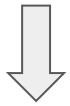
- **As we develop code for cleaning up HTML tags, we may also encounter various Unicode characters, including symbols, emojis, and other graphic characters**



Unicode normalization

- To parse such non-textual symbols and special characters, we use **Unicode normalization**, i.e., the text we see should be converted into some form of binary representation to store in a computer
  - When dealing with text in **multiple languages**, **social media data**, etc., we may have to convert between these encoding schemes during the text-extraction process

```
text = 'I love 🍕 !  Shall we book a 🚕 to gizza?'
Text = text.encode("utf-8")
print(Text)
```



```
b'I love Pizza \xf0\x9f\x8d\x95!  Shall we book a cab \xf0\x9f\x9a\x95
  to get pizza?'
```

Spelling correction

- **Due to fast typing and fat-finger typing [3], input text data often has spelling errors**
- This can be prevalent in **search engines**, **text-based chatbots** deployed on mobile devices, **social media**, and many other sources
- While we remove HTML tags and handle Unicode characters, this remains a unique **problem that may hurt the linguistic understanding of the data**

Shorthand typing: Hllo world! I am back!
Fat finger problem: I pronise that I will not bresk the silence again!

[3] https://en.wikipedia.org/wiki/Fat-finger_error

Spelling correction

- Despite our understanding of the problem, we don’t have a robust method to fix this, but there are some good solutions:
  - **Microsoft Bing Spell Check Rest API** [4]
  - **User implementation on Huggingface** [5]
  - **Pure Python Spell Checking library** [6]

- We can always implement our own solution ;)

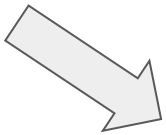
[4] https://learn.microsoft.com/en-us/azure/cognitive-services/bing-spell-check/quickstarts/python
[5] https://huggingface.co/oliverguhr/spelling-correction-english-base?text=lets+do+a+comparsion
[6] https://pypi.org/project/pyspellchecker/

Spelling correction (example)

```python
import requests
import json

api_key = "<ENTER-KEY-HERE>"
example_text = "Hollo, wrld" # the text to be spell-checked

data = {'text': example_text}
params = {
    'mkt':'en-us',
    'mode':'proof'
    }
headers = {
    'Content-Type': 'application/x-www-form-urlencoded',
    'Ocp-Apim-Subscription-Key': api_key,
    }
response = requests.post(endpoint, headers=headers, params=params, data=data)
json_response = response.json()
print(json.dumps(json_response, indent=4))
```



Output (partially shown here):

```
"suggestions": [
            {
                "suggestion": "Hello",
                "score": 0.9115257530801
            },
            {
                "suggestion": "Hollow",
                "score": 0.858039839213461
            },
            {
                "suggestion": "Hallo",
                "score": 0.597385084464481
            }
```

System-specific error correction - PDF documents

- Consider another scenario where our dataset is in the form of a collection of **PDF documents**
  - The pipeline in this case **starts with extraction of plain text from PDF documents**

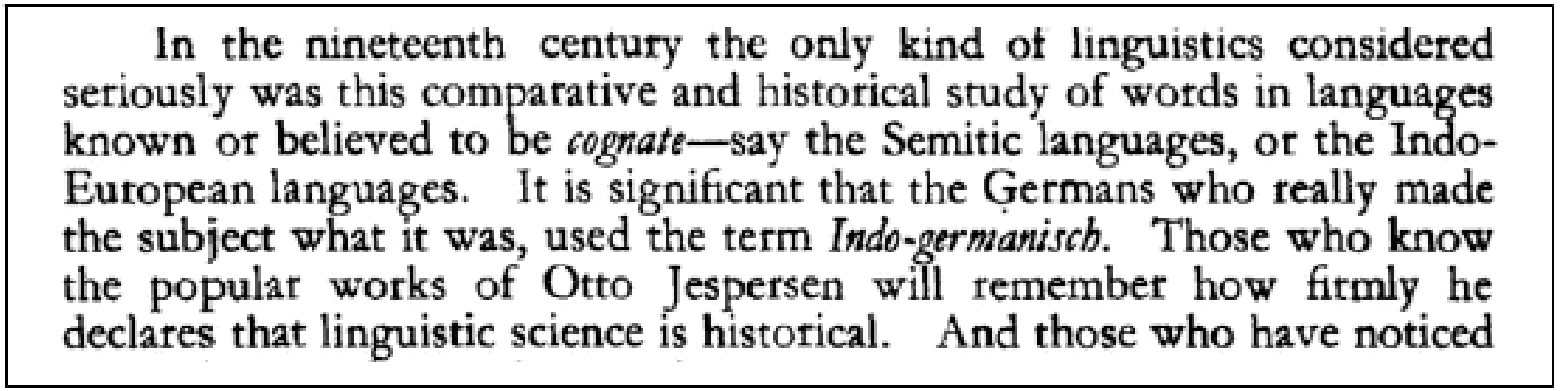
- Different PDF documents are encoded differently, and sometimes, we may not be able to extract the full text, or the structure of the text may get messed up - **this can impact our application!**
- There are several libraries: PyPDF4 [8], PDFMiner [9]
  - Far from perfect, not uncommon to encounter PDF documents that cannot be processed

[8] https://pypi.org/project/PyPDF4/
[9] https://pypi.org/project/pdfminer/

System-specific error correction - scanned documents

- **Another common source of textual data is scanned documents**

- Text extraction from scanned documents is typically done through **optical character recognition (OCR)**, using libraries such as **Tesseract** [10]

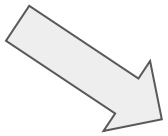


In the nineteenth century the only kind of linguistics considered seriously was this comparative and historical study of words in languages known or believed to be *cognate*—say the Semitic languages, or the Indo-European languages. It is significant that the Germans who really made the subject what it was, used the term *Indo-germanisch*. Those who know the popular works of Otto Jespersen will remember how firmly he declares that linguistic science is historical. And those who have noticed

[10] https://pypi.org/project/pytesseract/

System-specific error correction - scanned documents

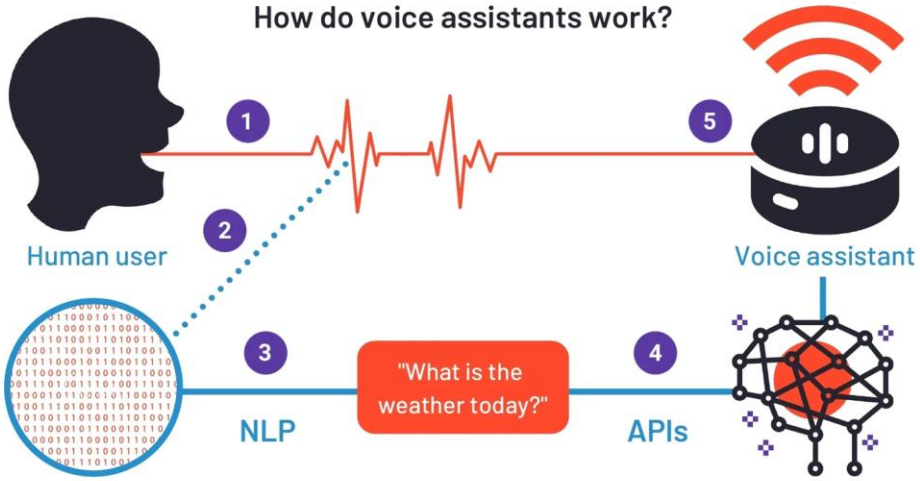
```
from PIL import Image
from pytesseract import image_to_string
filename = "somefile.png"
text = image_to_string(Image.open(filename))
print(text)
```



```
'in the nineteenth century the only Kind of linguistics considered\nseriously
was this comparative and historical study of words in languages\nknown or
believed to Fe cognate—say the Semitic languages, or the Indo-\nEuropean
languages. It is significant that the Germans who really made\nthe subject what
it was, used the term Indo-germanisch. Those who know\nthe popular works of
Otto Jespersen will remember how fitmly he\ndeclares that linguistic
science is historical. And those who have noticed'
```

System-specific error correction - speech

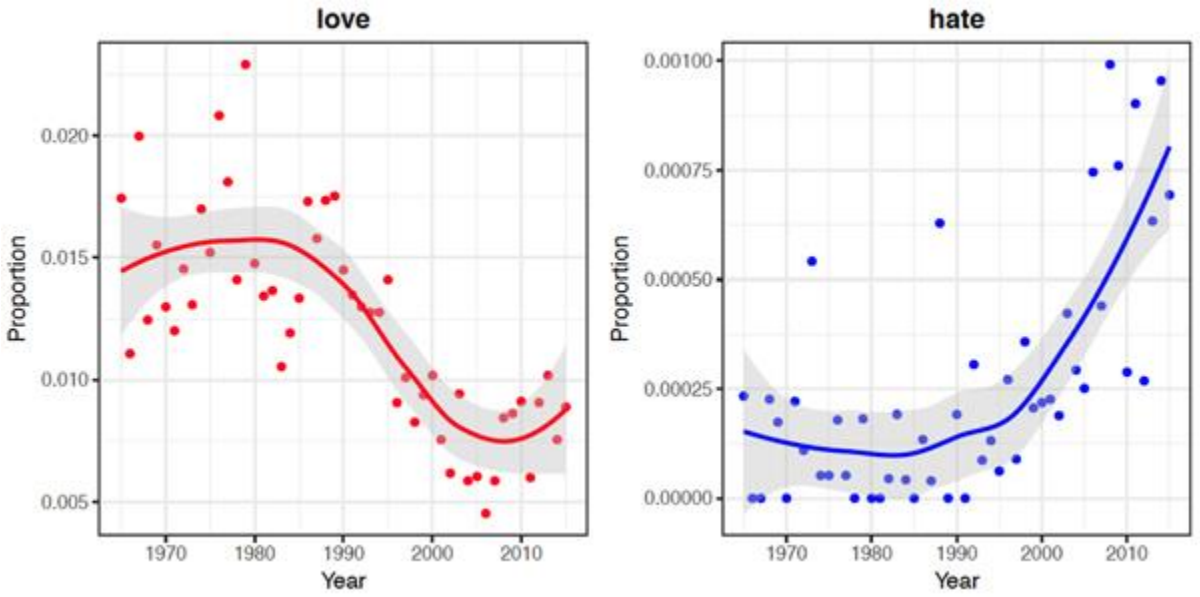
- **E.g., voice assistant extracts text from speech using automatic speech recognition (ASR)**
- Like OCR, it's common to see some errors in ASR, owing to various factors, such as dialectical variations, slang, non-native English, new or domain-specific vocabulary, etc.



[10] https://www.g2.com/articles/voice-assistant

System-specific error correction - speech

- E.g., **YouTube API** provides automatically created video transcripts and there are libraries for fetching music lyrics from **Spotify**



[11] Brand, C. O., Acerbi, A., & Mesoudi, A. (2019). Cultural evolution of emotional expression in 50 years of song lyrics. *Evolutionary Human Sciences*, *1*.

Pre-processing

Why pre-processing?

- **We already did some cleanup in the previous step, why do we still have to pre-process text?**

Why pre-processing?

- **We already did some cleanup in the previous step, why do we still have to pre-process text?**

- **All NLP software typically works at the sentence level and expects a separation of words at the minimum**
  - We need some way to **split a text into words and sentences** before proceeding further in a processing pipeline
- Sometimes, we need to **remove special characters and digits**, and sometimes, we don't care whether a word is in **upper** or **lowercase** and want everything in lowercase

- **Such decisions are addressed during the pre-processing step of the NLP pipeline**

Common pre-processing steps

1. **Preliminaries**
   - Sentence segmentation and word tokenization

2. **Frequent steps**
   - Stop word removal, stemming and lemmatization, removing digits/punctuation, lowercasing, etc.

3. **Other steps**
   - Normalization, language detection, code mixing, transliteration, etc.

4. **Advanced processing**
   - POS tagging, parsing, coreference resolution, etc.

Preliminaries

- **Any NLP pipeline has to start with a reliable system to split the text into sentences (sentence segmentation) and further split a sentence into words (word tokenization)**

- **On the surface, these seem like simple tasks:**
  - As a simple rule, we can do sentence segmentation by breaking up text into sentences at the appearance of full stops and question marks. However, there may be abbreviations, forms of addresses (Dr., Mr., etc.), or ellipses (...) that may break the simple rule

- Thankfully, we don't have to worry about how to solve these issues, as most NLP libraries come with some form of sentence and word splitting implemented, e.g., **Natural Language Tool Kit (NLTK)** [12]

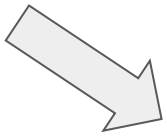
[12] https://www.nltk.org/

Sentence segmentation

```
from nltk.tokenize import sent_tokenize, word_tokenize

mytext = "In the previous chapter, we saw examples of some common NLP
applications that we might encounter in everyday life. If we were asked to
build such an application, think about how we would approach doing so at our
organization. We would normally walk through the requirements and break the
problem down into several sub-problems, then try to develop a step-by-step
procedure to solve them. Since language processing is involved, we would also
list all the forms of text processing needed at each step. This step-by-step
processing of text is known as pipeline. It is the series of steps involved in
building any NLP model. These steps are common in every NLP project, so it
makes sense to study them in this chapter. Understanding some common procedures
in any NLP pipeline will enable us to get started on any NLP problem encountered
in the workplace. Laying out and developing a text-processing pipeline is seen
as a starting point for any NLP application development process. In this
chapter, we will learn about the various steps involved and how they play
important roles in solving the NLP problem and we'll see a few guidelines
about when and how to use which step. In later chapters, we'll discuss
specific pipelines for various NLP tasks (e.g., Chapters 4-7)."

my_sentences = sent_tokenize(mytext)
```

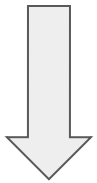


```
In the previous chapter, we saw a quick overview of what is NLP, what are some
of the common applications and challenges in NLP, and an introduction to
different tasks in NLP.
```

Word tokenization

```
print(word_tokenize(sentence))
```

```
In the previous chapter, we saw a quick overview of what is NLP, what are some
of the common applications and challenges in NLP, and an introduction to
different tasks in NLP.
```



```
['In', 'the', 'previous', 'chapter', ',', 'we', 'saw', 'a', 'quick',
'overview', 'of', 'what', 'is', 'NLP', ',', 'what', 'are', 'some', 'of', 'the',
'common', 'applications', 'and', 'challenges', 'in', 'NLP', ',', 'and', 'an',
'introduction', 'to', 'different', 'tasks', 'in', 'NLP', '.']
```

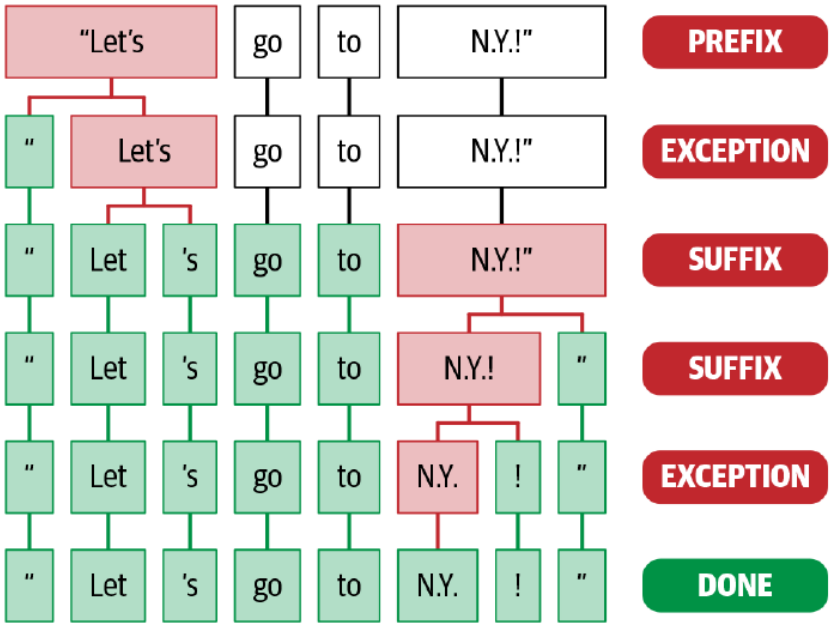
Word tokenization

- Available word tokenizers are still far from perfect
- Basic word tokenizer would separate a #twitter into two separate tokens
- Social media such as Twitter uses a specific language and requires specialized tokenizers
  - NLTK also offers a dedicated tweet tokenizer [13]
- Tokenization can also differ from one language to another
  - Each language may have various linguistic rules and exceptions

[13] https://www.nltk.org/api/nltk.tokenize.casual.html

Word tokenization

- Such language-specific exceptions can be specified in the tokenizer provided by spaCy [14]
  - It’s also possible in spaCy to develop custom rules



[13] https://spacy.io/

Word tokenization

- **Sentence segmenters and word tokenizers are sensitive to the input they receive:**
  - Let's say we're writing software to extract some information, such as company, position, and salary, from job offer letters which follow a certain format
    - How will we decide what a sentence is in such a case?
    - Should the entire address be considered a single "sentence"?
    - Or should each line be split separately?
    - Answers to such questions depend on what you want to extract and how sensitive the rest of the pipeline is about such decisions

- For identifying specific patterns (e.g., dates or money expressions), well-formed **regular expressions** are the first step
- In many practical scenarios, we may end up using a custom tokenizer or sentence segmenter that suits our text structure instead of or on top of an existing one available in a standard NLP library

Regular expressions

- **A formal language for specifying text strings**
- Regular expressions play a surprisingly large role
  - Sophisticated sequences of regular expressions are often the first model for any text processing text

- How can we search for any of these?
  - **woodchuck**
  - **woodchucks**
  - **Woodchuck**
  - **Woodchucks**

Regular expressions: disjunctions

- Letters inside square brackets **[ ]**

Pattern	Matches
`[wW]oodchuck`	Woodchuck, woodchuck
`[1234567890]`	Any digit

- Ranges **[A-Z]**

Pattern	Matches	
`[A-Z]`	An upper case letter	`Drenched Blossoms`
`[a-z]`	A lower case letter	`my beans were impatient`
`[0-9]`	A single digit	`Chapter 1: Down the Rabbit Hole`

Regular expressions: negation in disjunction

- Negations **[^Ss]**
  - Carat means negation only when first in []

Pattern	Matches	
[^A-Z]	Not an upper case letter	Oyfn pripetchik
[^Ss]	Neither 'S' nor 's'	I have no exquisite reason"
[^e^]	Neither e nor ^	Look here
a^b	The pattern a carat b	Look up a^b now

Regular expressions: more disjunction

- Woodchuck is another name for groundhog!
- The **pipe |** for disjunction

Pattern	Matches
`groundhog|woodchuck`	
`yours|mine`	yours mine
`a|b|c`	= [abc]
`[gG]roundhog|[Ww]oodchuck`	

Regular expressions: ? * + .

Pattern	Matches	
colou?r	Optional previous char	color colour
oo*h!	0 or more of previous char	oh! ooh! oooh! ooooh!
o+h!	1 or more of previous char	oh! ooh! oooh! ooooh!
baa+		baa baaa baaaa baaaaa
beg.n		begin begun begun beg3n

Regular expressions: anchors ^ $

Pattern	Matches
^[A-Z]	Palo Alto
^[^A-Za-z]	1 "Hello"
\.$	The end.
.$	The end? The end!

Frequent steps

- **We’re designing software that identifies the category of a news article as one of politics, sports, business, and other**
  - Assume we have a good sentence segmenter and word tokenizer in place
  - Now we need to start thinking about what kind of information is useful for developing a categorization tool

Frequent steps: stop words removal

- Some of the frequently used words in English, such as **a**, **an**, **the**, **of**, **in**, etc., are not particularly useful for this task
  - They don’t carry any content on their own to separate between the categories
- Such words are called **stop words** and are typically removed from further analysis in such problem scenarios
- No standard list for English, there are some popular lists (in NLTK)
- What a stop word is **can vary on the context**
  - E.g., the word **news** is perhaps a stop word in this scenario

Frequent steps: lowercasing/uppercasing, punctuation/digit removal

- **Upper** or **lower case** may often not make a difference for the problem, so **all text is lowercased** (or uppercased, although lowercasing is more common)
- **Removing punctuation and/or numbers** is also a common step for many NLP problems

```
from nltk.corpus import stopwords
From string import punctuation
def preprocess_corpus(texts):
    mystopwords = set(stopwords.words("english"))
    def remove_stops_digits(tokens):
       return [token.lower() for token in tokens if token not in mystopwords
               not token.isdigit() and token not in punctuation]
    return [remove_stops_digits(word_tokenize(text)) for text in texts]
```

Frequent steps

- Stop words removal, lowercasing/uppercasing, punctuation/digit removal **are not mandatory or sequential** in nature
  - Also, **this type of pre-processing**, while specific to textual data has **nothing particularly linguistic** about it
  - Stop words are simply very frequent words
  - And we're removing non-alphabetic data (punctuation, digits)

- Two commonly used pre-processing steps that take the word-level properties into account are **stemming** and **lemmatization**

Frequent steps: stemming and lemmatization

- **Stemming** refers to the process of removing suffixes and reducing a word to some base form such that all different variants of that word can be represented by the same form
  - E.g., ¸'car' and 'cars' are both reduced to 'car'
- This is accomplished by applying a **fixed set of rules**
  - E.g., if the word ends in '-es', remove '-es'

- Although such rules may not always end up in a linguistically correct base form, stemming is commonly used in search engines to match user queries to relevant documents and in text classification to reduce the feature space to train machine learning models

Frequent steps: stemming

```
from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()
word1, word2 = “cars”, “revolution”
print(stemmer.stem(word1), stemmer.stem(word2))
```

- Returns **‘car’** as the stemmed version for **‘cars’**, but **‘revolut’** as the stemmed form of **‘revolution’** even though the latter is **not linguistically correct**

- However, in some scenarios, derivation of correct linguistic form may become useful → **lemmatization**

Frequent steps: lemmatization

- **Lemmatization** is the process of mapping all the different forms of a word to its base word, or **lemma**
- Seems similar to stemming, but they are very different
  - E.g., adjective **'better'** when stemmed, remains the same, but upon lemmatization it becomes **'good'**
- **Lemmatization requires more linguistic knowledge**, and modeling and developing efficient lemmatizers remains an open problem in NLP research even now

Stemming	Lemmatization
adjustable -> adjust	was -> (to) be
formality -> formaliti	better -> good
formaliti -> formal	meeting -> meeting
airliner -> airlin	

Frequent steps: lemmatization

- Lemmatizer based on WordNet from NLTK:

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordnetLemmatizer()
print(lemmatizer.lemmatize("better", pos="a")) #a is for adjective
```

- Lemmatizer using spaCy:

```
import spacy
sp = spacy.load('en_core_web_sm')
token = sp(u'better')
for word in token:
   print(word.text,  word.lemma_)
```

Frequent steps: lemmatization

- NLTK prints the output as "**good**", whereas spaCy prints "**well**" – both are correct

- Since lemmatization involves some amount of linguistic analysis of the word and its context, it is expected that it will take longer to run than stemming, and it's also **typically used only if absolutely necessary**

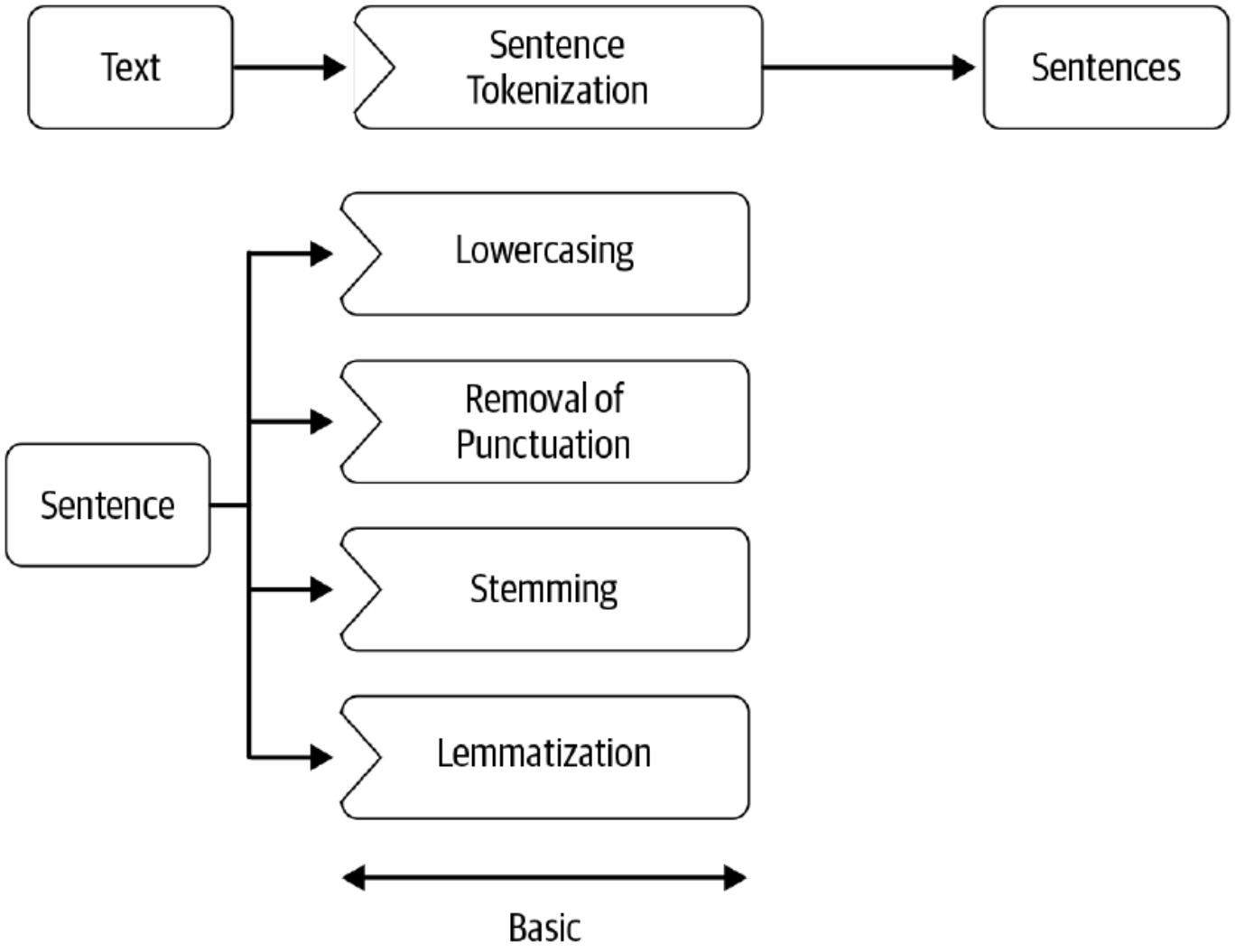
- **The choice of lemmatizer is optional**; we can choose NLTK or spaCy given what framework we're using for other pre-processing steps in order to use a single framework in the complete pipeline

Stemming vs Lemmatization

Frequent steps

- **Not all of these steps are always necessary** and not all of them are performed in this order
- Typically **lowercasing is performed before stemming**
- Also **don’t remove the tokens or lowercase the text before doing lemmatization**
  - Because we have to know the part of speech of the word to get its lemma, and that requires all tokens in the sentence to be intact
- A good practice to follow is to **prepare a sequential list of pre-processing tasks** to be done after having a clear understanding of how to process our data

Common pre-processing steps



Other pre-processing steps

- So far, we've assumed that we're dealing with **regular English text**
  - What's different if that's not the case?

Text normalization

- Consider a scenario where we're working with a collection of social media posts to detect news events
- Social media text is **very different** from news language
- It's useful to reach a canonical representation of text that captures all these variations into one representation
  - This is known as **text normalization**
- E.g., convert all text to uppercase or lowercase, convert digits to text (e.g., 9 to nine), expand abbreviations, replace emojis with text, etc.

Language detection

- A lot of web content is in **non-English languages**
- If our **NLP pipeline is built to expect English** text as input, it will not be able to properly deal with such content
- In such cases, **language detection** is performed as the **first step** in an NLP pipeline (e.g., using Polyglot [14])
- Once translated, next steps could follow a language-specific pipeline

[14] https://polyglot.readthedocs.io/en/latest/

Code mixing and transliteration

- What if a single piece of content is in **more than one language**?
- **Not uncommon** as many people across the world speak more than one language in their daily lives
  - Especially in their **social media** posts

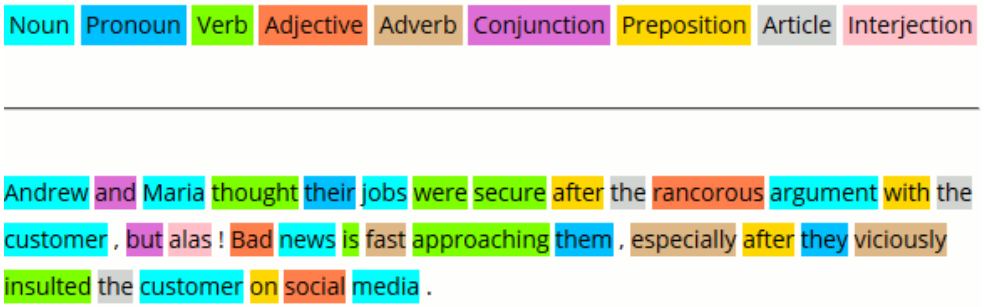
Translation: Hey, when we date we always eat at the coffeeshop (one).

Code mixing and transliteration

- A single popular phrase has words from Tamil, English, Malay, and three Chinese language variants
- **Code mixing** refers to this phenomenon of switching between languages
- When people use multiple languages in their write-ups, they often type words from these languages in Roman script, with English spelling
  - So, the words of another language are written along with English text
- This is known as **transliteration**
- Both of these phenomena are common and need to be handled during pre-processing

Advanced processing: POS tagging

- **Imagine we're asked to develop a system to identify person and organization names in our company's collection of one million documents**
- The common pre-processing steps steps we discussed earlier may not be relevant in this context
- Identifying names requires us to be able to do **POS (part-of-speech) tagging**, as identifying proper nouns can be useful in identifying person and organization names



Advanced processing: POS tagging

- **Pre-trained and readily usable POS taggers are implemented in NLTK and SpaCy, and we generally don’t have to develop our own POS-tagging solutions**

```
import spacy
nlp = spacy.load('en_core_web_sm')
doc = nlp(u'Charles Spencer Chaplin was born on 16 April 1889 toHannah Chaplin
         (born Hannah Harriet
Pedlingham Hill) and Charles Chaplin Sr')
for token in doc:
    print(token.text, token.lemma_, token.pos_,
          token.shape_, token.is_alpha, token.is_stop)
```

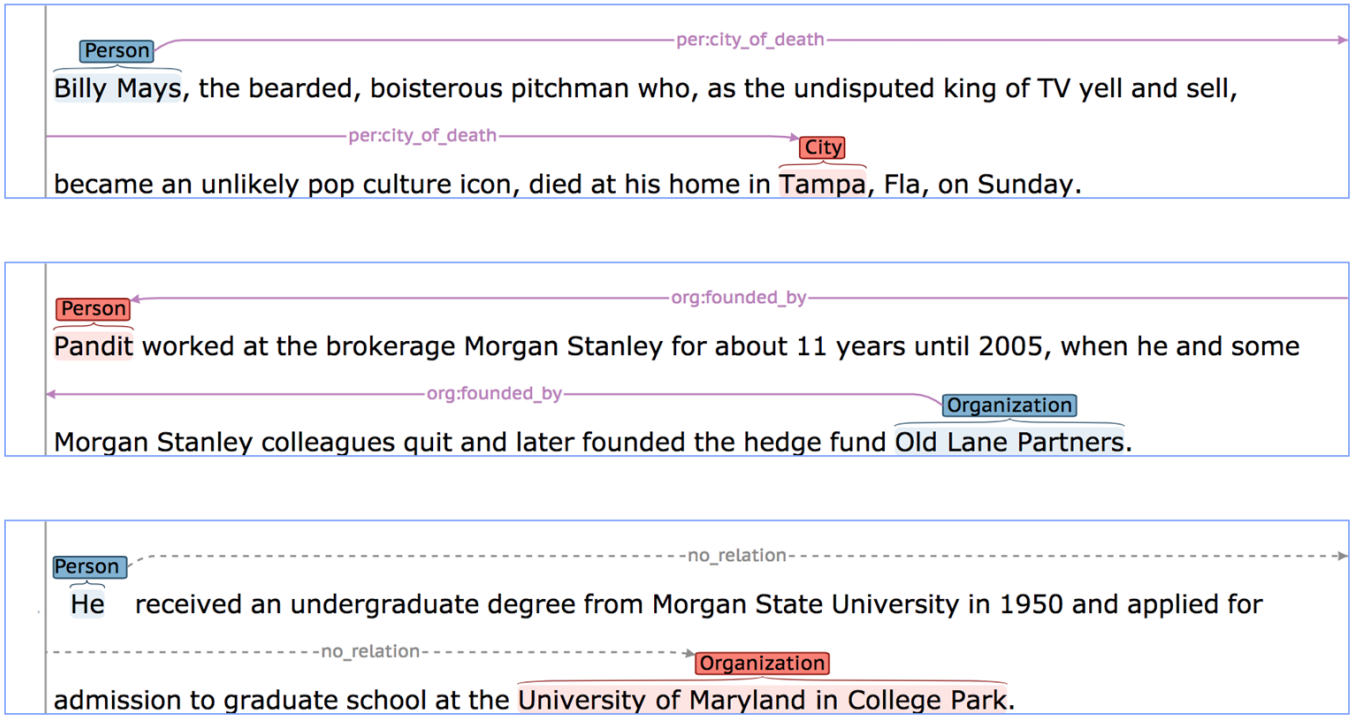
Advanced processing: POS tagging

- **Why is POS tagging useful?**
  - POS tagging can be really useful, particularly if you have words or tokens that can have multiple POS tags. For instance, the word "google" can be used as both a noun and verb, depending upon the context
  - While processing natural language, it is important to identify this difference

- The SpaCy library comes **pre-built with machine learning algorithms** that, depending upon the context (surrounding words), it is capable of returning the correct POS tag for the word

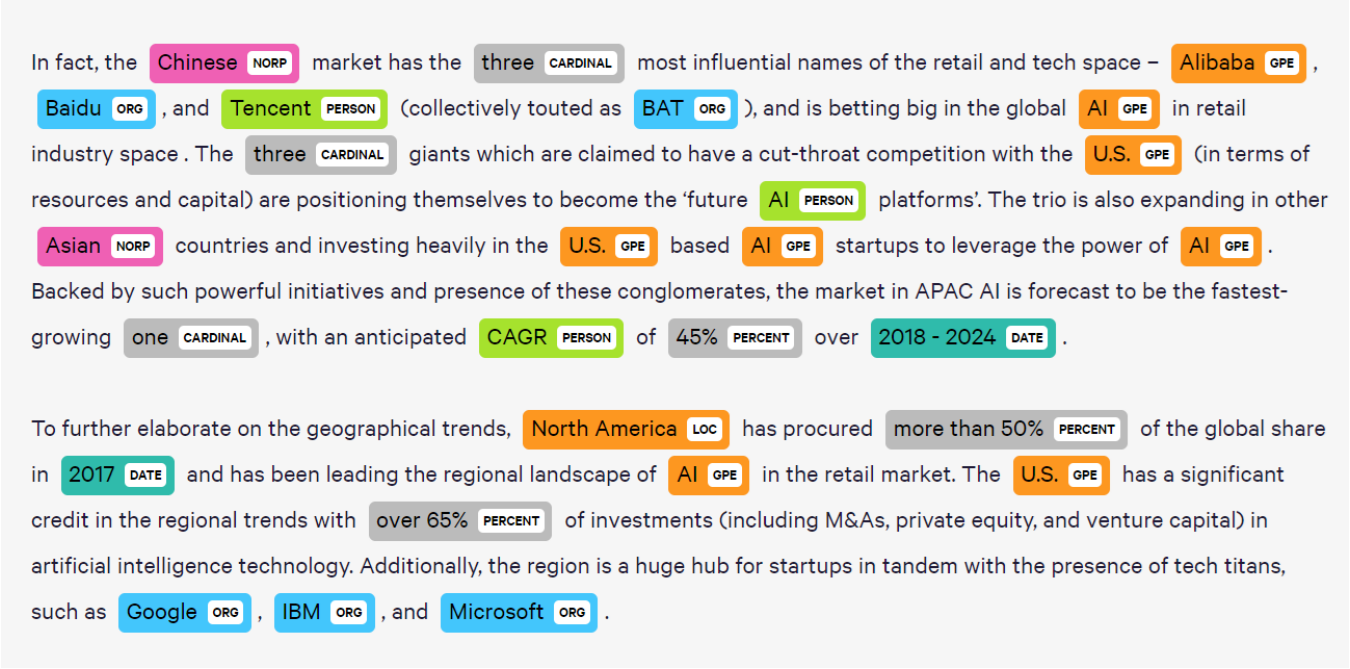
Advanced processing: relation extraction

- Identifying if a given **person** and **organization are related to each other** in some way



Advanced processing: named entity recognition (NER)

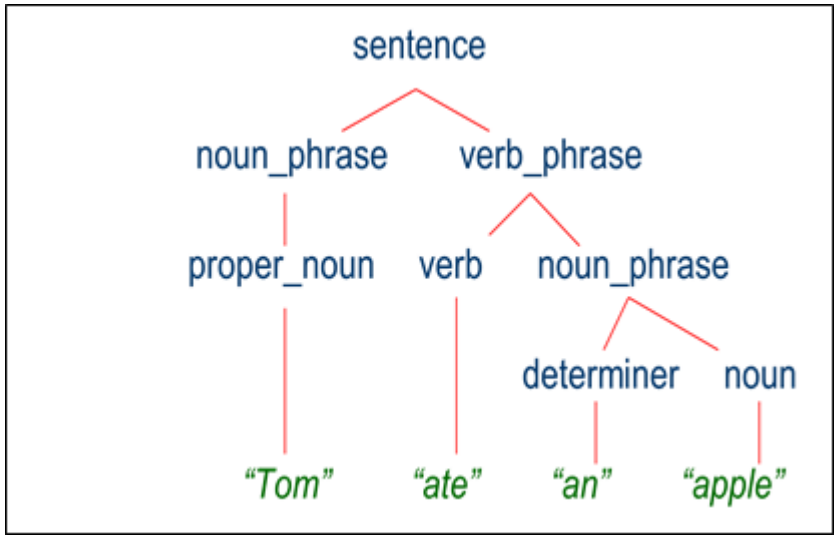
- Subtask of information extraction that seeks to **locate** and **classify named entities** mentioned in unstructured text into pre-defined categories such as **person names**, **organizations**, **locations**, **medical codes**, **time expressions**, **quantities**, **monetary values**, **percentages**, etc.



Advanced processing: parsing

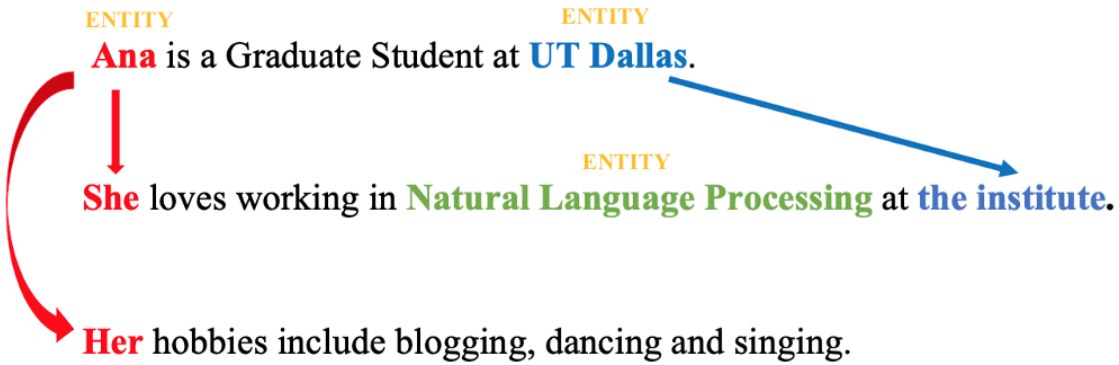
- If we need to identify patterns indicating **‘relation’** between two entities in a sentence, we need to have some form of **syntactic representation** of the sentence, such as **parsing**
  - Process of determining the syntactic structure of a text by analyzing its constituent words based on an underlying grammar (of the language)

```
sentence -> noun_phrase, verb_phrase
noun_phrase -> proper_noun
noun_phrase -> determiner, noun
verb_phrase -> verb, noun_phrase
proper_noun -> [Tom]
noun -> [apple]
verb -> [ate]
determiner -> [an]
```

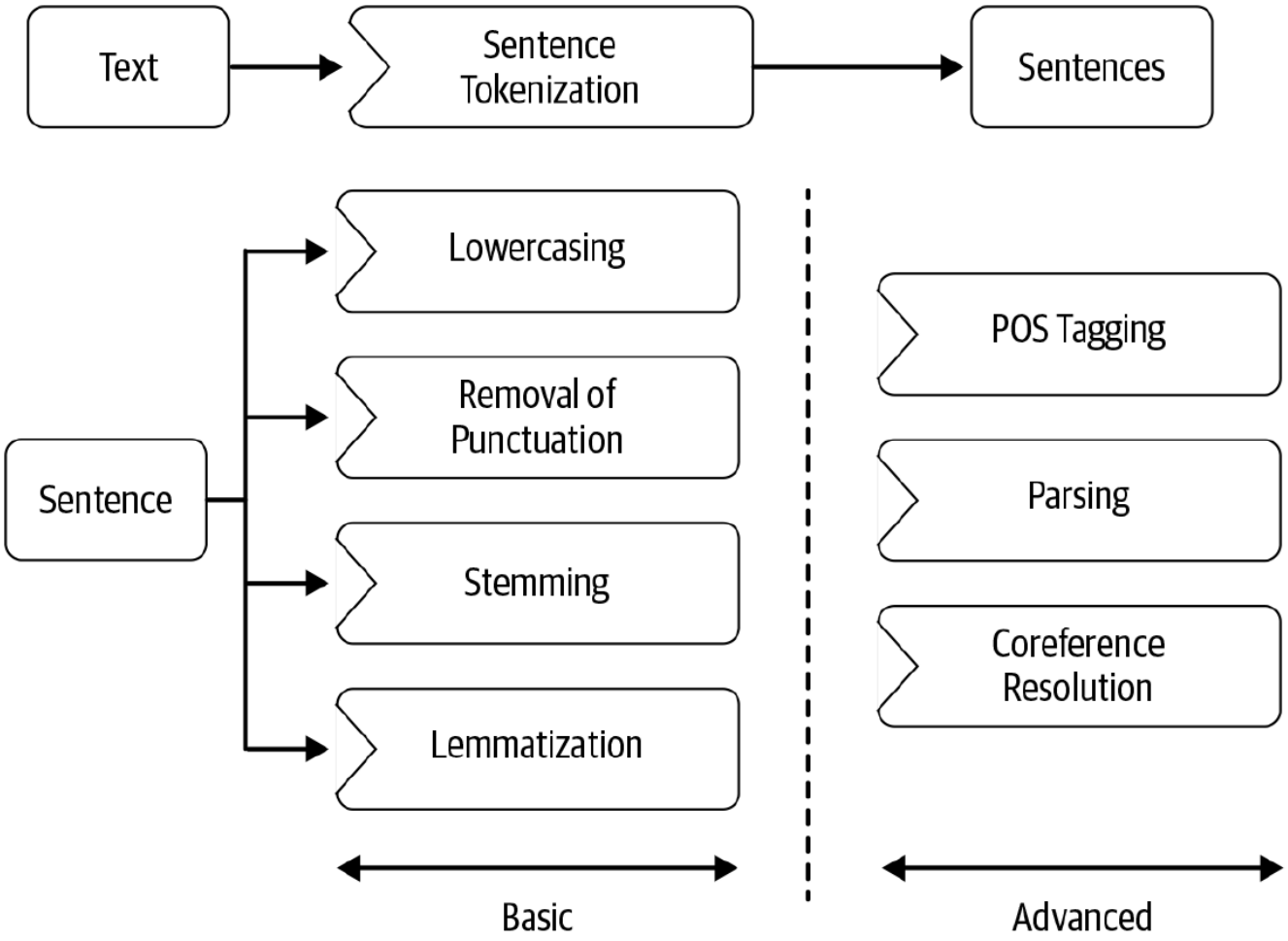


Advanced processing: coreference resolution

- If we want a way to **identify** and **link multiple mentions of an entity** (e.g., Satya Nadella, Mr. Nadella, he, etc.)



Common pre-processing steps



Ordering of the pre-processing steps is important

- E.g., POS tagging cannot be preceded by stop word removal, lowercasing, etc.
  - Such processing affects POS tagger output by changing the grammatical structure of the sentence

- How a particular pre-processing step is helping a given NLP problem is another question that is specific to the application, and it can only be answered with a lot of experimentation

Hands-on

- Text cleaning and preprocessing: https://shorturl.at/GmPIG
- Regular expressions: https://shorturl.at/gVfGs
- Download image: https://ibb.co/k8R0Wgp

What we've learned today

- Character Encoding:
  - Importance in digital representation
  - Unicode's versatility
- Regular expressions
- NLP Pre-processing:
  - Conditional steps application
  - Sequence importance (e.g., lowercasing before stemming)
  - Context necessity in lemmatization
- Advanced Processing
- Strategic Approach:
  - Tailored pre-processing pipelines
  - Data-specific strategies

Today's session highlighted the intricacies of NLP, emphasizing strategic, data-specific approaches for effective pre-processing.