

Natural Language Processing

- Computers are great at working with structured data like spreadsheets and database tables. But us humans usually communicate in words, not in tables. That's unfortunate for computers.
- A lot of information in the world is unstructured — raw text in English or another human language. How can we get a computer to understand unstructured text and extract data from it?

London is the capital and most populous city of England and the United Kingdom. Standing on the River Thames in the south east of the island of Great Britain, London has been a major settlement for two millennia. It was founded by the Romans, who named it Londinium. London's ancient core, the City of London, largely retains its 1.12-square-mile (2.9 km²) medieval boundaries.

Natural Language Processing, or ***NLP***, is the sub-field of AI that is focused on enabling computers to **understand** and **process human languages**. We will see how NLP works and learn how to write programs that can extract information out of raw text using Python!

- Computers can't yet truly understand English in the way that humans do — but they can already do a lot! In certain limited areas, what we can do with NLP already seems like magic. We might be able to save a lot of time by applying NLP techniques to our own projects.
- And even better, the latest advances in NLP are easily accessible through open source Python libraries like **spaCy**, **textacy**, and **neuralcoref**. What you can do with just a few lines of python is amazing.
- I have attached the links to their official documentation which you can use and read along to know everything about them in detail:
 1. <https://spacy.io/>
 2. <https://textacy.readthedocs.io/en/latest/>
 3. <https://github.com/huggingface/neuralcoref>
- Ok! so now let's see what they are and how to work with them...

Extracting Meaning from Text is Hard

- The process of reading and understanding English is very complex — and that's not even considering that English doesn't follow logical and consistent rules. For example, what does this news headline mean?

“Environmental regulators grill business owner over illegal coal fires.”

- Are the regulators questioning a business owner about burning coal illegally? Or are the regulators literally cooking the business owner? As you can see, parsing English with a computer is going to be complicated.
- Doing anything complicated in machine learning usually means **building a pipeline**. The idea is to break up our problem into **very small pieces** and then use machine learning to solve each smaller piece separately. Then by **chaining together several machine learning models** that **feed into each other**, we can do very complicated things.

Building an NLP Pipeline, Step-by-Step

Let's look at a piece of text from Wikipedia:

London is the capital and most populous city of England and the United Kingdom. Standing on the River Thames in the south east of the island of Great Britain, London has been a major settlement for two millennia. It was founded by the Romans, who named it Londinium.

This paragraph contains several useful facts. It would be great if a computer could read this text and understand that **London is a city, London is located in England, London was settled by Romans** and so on.

But to get there, we have to first teach our computer the most basic concepts of written language and then move up from there.

Let's see how we are going to do this!

Step 1: Sentence Segmentation

The first step in the pipeline is to **break the text apart into separate sentences**. That gives us this:

1. “London is the capital and most populous city of England and the United Kingdom.”
2. “Standing on the River Thames in the south east of the island of Great Britain, London has been a major settlement for two millennia.”
3. “It was founded by the Romans, who named it Londinium.”

We can assume that each sentence in English is a separate thought or idea. It will be a lot easier to write a program to understand a single sentence than to understand a whole paragraph.

Coding a Sentence Segmentation model can be as simple as splitting apart sentences whenever you see a **punctuation mark**. But modern NLP pipelines often use more complex techniques that work **even when a document isn't formatted cleanly**.

Step 2: Word Tokenization

Now that we've split our document into sentences, we can process them one at a time. Let's start with the first sentence from our document:

“London is the capital and most populous city of England and the United Kingdom.”

The next step in our pipeline is to break this sentence into separate words or **tokens**. This is called **tokenization**. This is the result:

“London”, “is”, “the”, “capital”, “and”, “most”, “populous”, “city”, “of”, “England”, “and”, “the”, “United”, “Kingdom”, “.”

Tokenization is easy to do in English. We'll just split apart words whenever there's a **space** between them. And we'll also treat **punctuation marks as separate tokens** since punctuation also has meaning.

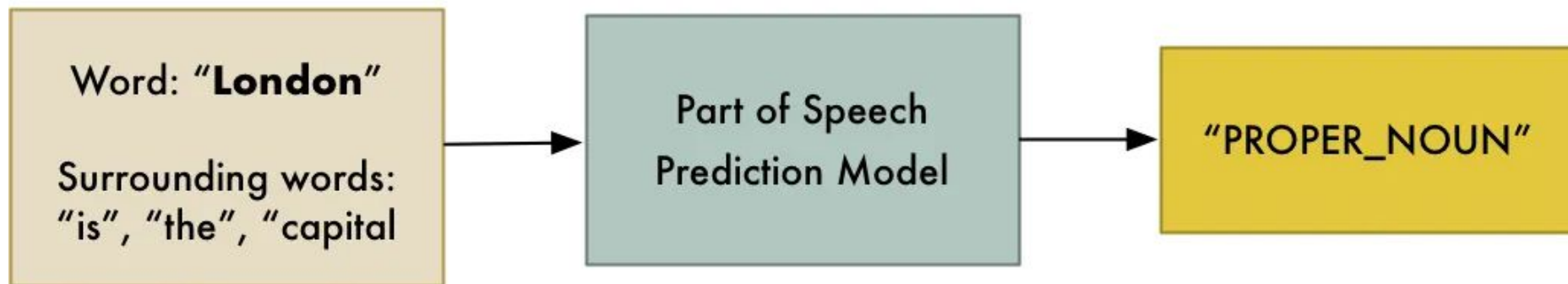
Step 3: Predicting Parts of Speech for Each Token

Next, we'll look at each token and try to guess its part of speech — whether it is a noun, a verb, an adjective and so on. Knowing the role of each word in the sentence will help us start to figure out what the sentence is talking about.

We can do this by feeding each word (and some extra words around it for context) into a pre-trained part-of-speech classification model:

Input

Output



The part-of-speech model was originally trained by feeding it millions of English sentences with each word's part of speech already tagged and having it learn to replicate that behavior.

Keep in mind that the model is completely based on statistics — it doesn't actually understand what the words mean in the **same way that humans do**. It just knows how to guess a part of speech based on similar **sentences and words it has seen before**.

After processing the whole sentence, we'll have a result like this:



With this information, we can already start to glean some very basic meaning. For example, we can see that the nouns in the sentence include “**London**” and “**capital**”, so the sentence is probably talking about London.

Step 4: Text Lemmatization

In English (and most languages), words appear in different forms. Look at these two sentences:

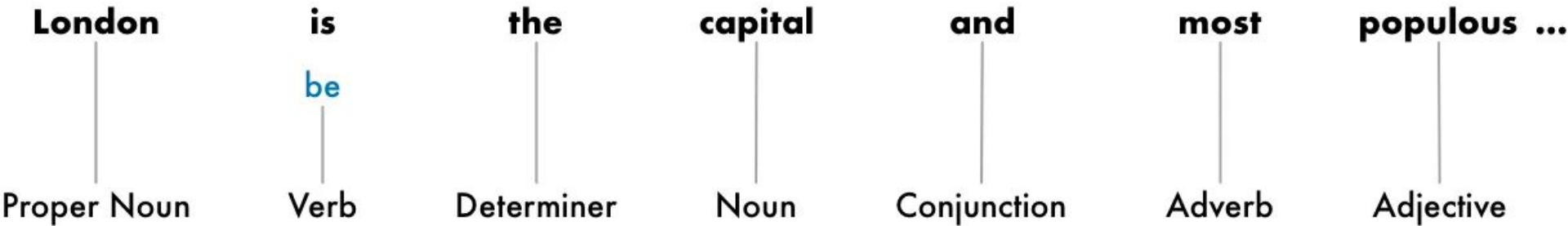
I had a **pony**.

I had two **ponies**.

Both sentences talk about the noun **pony**, but they are using different inflections. When working with text in a computer, it is helpful to know the base form of each word so that you know that both sentences are talking about the same concept. Otherwise the strings “**pony**” and “**ponies**” look like two totally different words to a computer.

In NLP, we call finding this process **lemmatization** — figuring out the most basic form or *lemma* of each word in the sentence. The same thing applies to verbs. We can also lemmatize verbs by finding their root, unconjugated form. So “**I had two ponies**” becomes “**I [have] two [pony].**”

Lemmatization is typically done by having a look-up table of the lemma forms of words based on their part of speech and possibly having some custom rules to handle words that you’ve never seen before. Here’s what our sentence looks like after lemmatization adds in the root form of our verb:



The only change we made was turning “is” into “be”.

Step 5: Identifying Stop Words

Next, we want to consider the importance of each word in the sentence. English has a lot of filler words that appear very frequently like “and”, “the”, and “a”. When doing statistics on text, these words introduce a lot of noise since they appear way more frequently than other words. Some NLP pipelines will flag them as **stop words**—that is, words that you might want to filter out before doing any statistical analysis. Here’s how our sentence looks with the stop words grayed out:

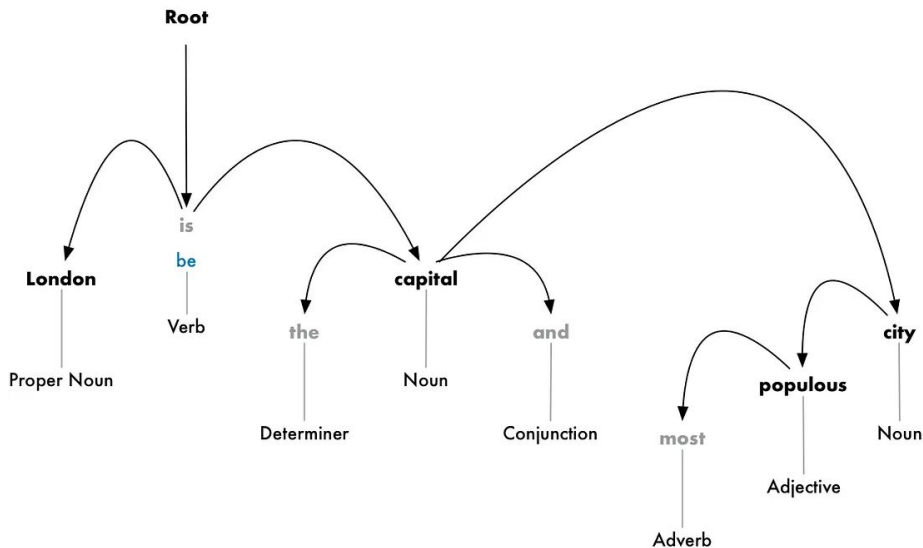


Stop words are usually identified by just by checking a hardcoded list of known stop words. But there’s no standard list of stop words that is appropriate for all applications. The list of words to ignore can vary depending on your application.

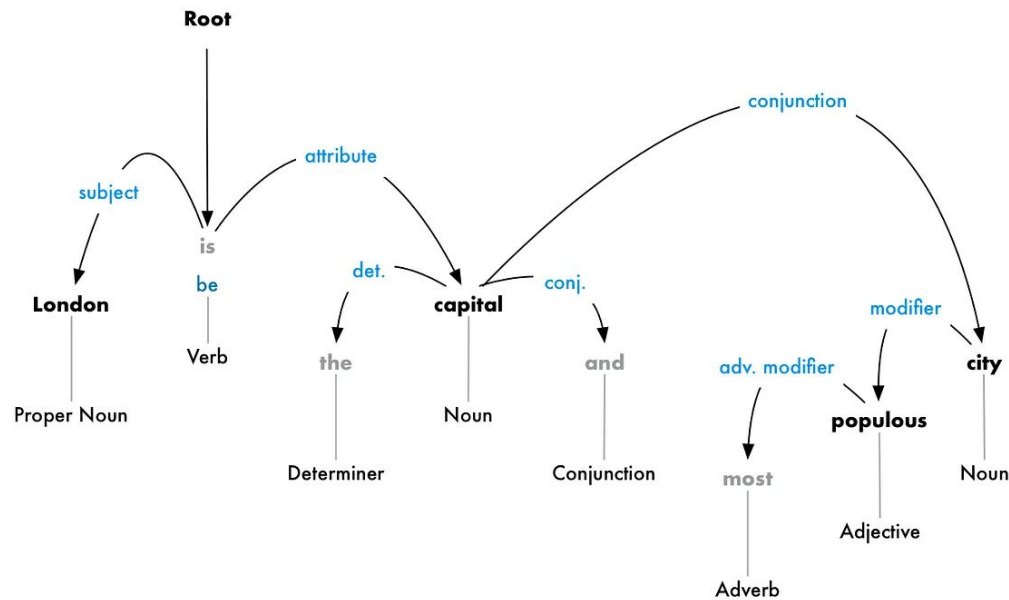
Step 6: Dependency Parsing

The next step is to figure out how all the words in a sentence relate to each other. This is called *dependency parsing*.

The goal is to build a tree that assigns a single **parent** word to each word in the sentence. The root of the tree will be the main verb in the sentence. Here's what the beginning of the parse tree will look like for our sentence:



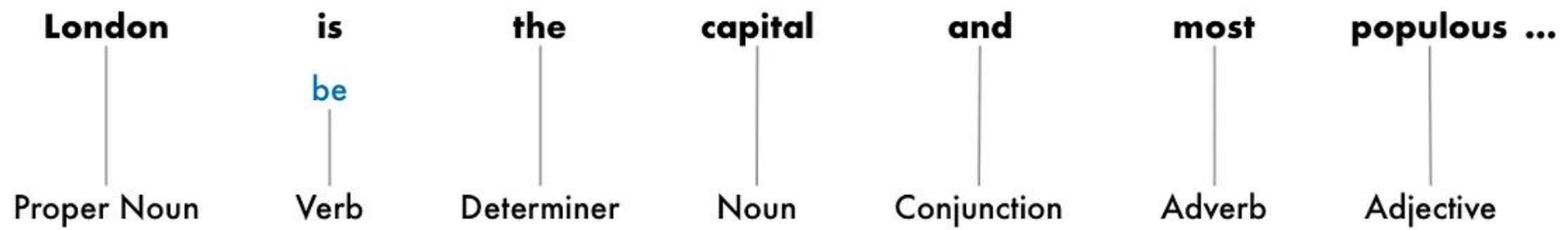
But we can go one step further. In addition to identifying the parent word of each word, we can also predict the type of relationship that exists between those two words:



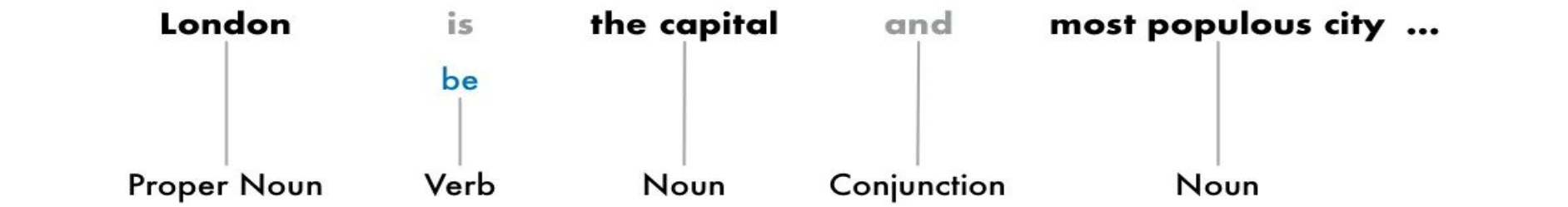
This parse tree shows us that the subject of the sentence is the noun “**London**” and it has a “**be**” relationship with “**capital**”. We finally know something useful — **London is a capital!** And if we followed the complete parse tree for the sentence (beyond what is shown), we would even found out that London is the capital of the *United Kingdom*.

Step 6b: Finding Noun Phrases

So far, we've treated every word in our sentence as a separate entity. But sometimes it makes more sense to group together the words that represent a single idea or thing. We can use the information from the dependency parse tree to automatically group together words that are all talking about the same thing. For example, instead of this:



We can group the noun phrases to generate this:



Step 7: Named Entity Recognition (NER)

Now that we've done all that hard work, we can finally move beyond grade-school grammar and start actually extracting ideas. In our sentence, we have the following nouns:

London is the **capital** and most populous **city** of **England** and the **United Kingdom**.

- Some of these nouns present real things in the world. For example, “*London*”, “*England*” and “*United Kingdom*” represent physical places on a map. It would be nice to be able to detect that! With that information, we could automatically extract a list of real-world places mentioned in a document using NLP.
- The goal of ***Named Entity Recognition, or NER***, is to detect and label these nouns with the real-world concepts that they represent. Here's what our sentence looks like after running each token through our NER tagging model:

London is the capital and most populous city of **England** and the **United Kingdom**.

Geographic
Entity

Geographic
Entity

Geographic
Entity

But NER systems aren't just doing a simple dictionary lookup. Instead, they are using the context of how a word appears in the sentence and a statistical model to guess which type of noun a word represents.

Here are just some of the kinds of objects that a typical NER system can tag:

- People's names
- Company names
- Geographic locations (Both physical and political)
- Product names
- Dates and times
- Amounts of money
- Names of events

Step 8: Coreference Resolution

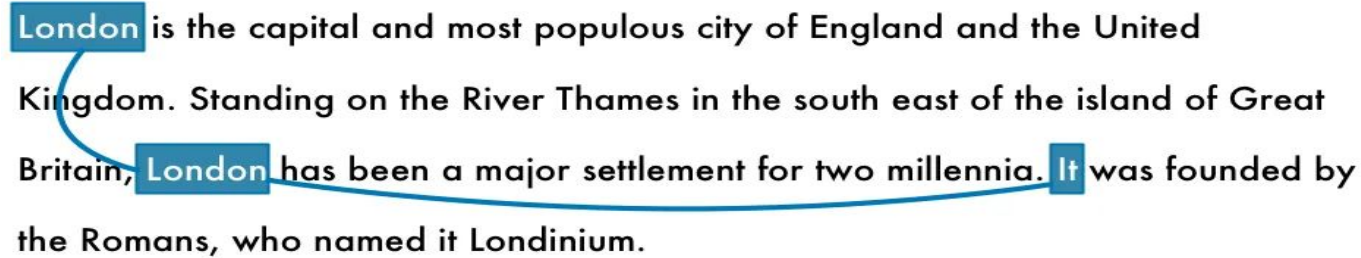
At this point, we already have a useful representation of our sentence. We know the parts of speech for each word, how the words relate to each other and which words are talking about named entities.

However, we still have one big problem. English is full of pronouns — words like *he*, *she*, and *it*. These are shortcuts that we use instead of writing out names over and over in each sentence. Humans can keep track of what these words represent based on context. But our **NLP model doesn't know what pronouns mean** because it only examines one sentence at a time. Let's look at the third sentence in our document:

“It was founded by the Romans, who named it Londinium.”

If we parse this with our NLP pipeline, we'll know that “**it**” was founded by Romans. But it's a lot more useful to know that “**London**” was founded by Romans.

As a human reading this sentence, you can easily figure out that “*it*” means “**London**”. The goal of coreference resolution is to figure out this same mapping by tracking pronouns across sentences. We want to figure out all the words that are referring to the same entity. Here’s the result of running coreference resolution on our document for the word “**London**”:



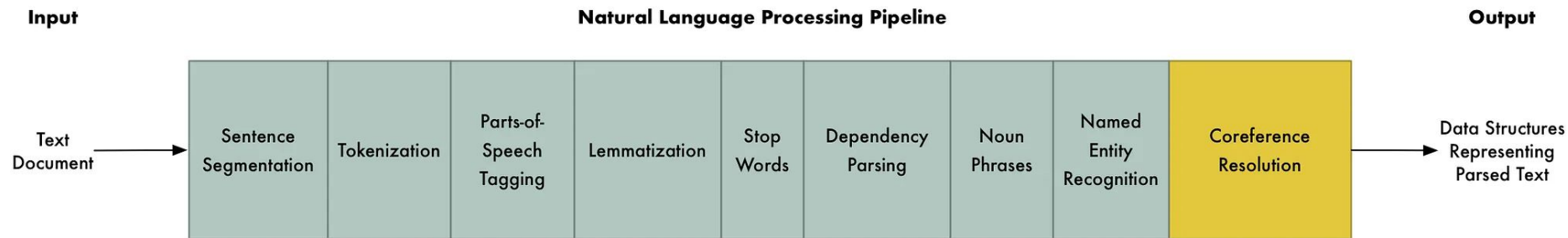
London is the capital and most populous city of England and the United Kingdom. Standing on the River Thames in the south east of the island of Great Britain, London has been a major settlement for two millennia. It was founded by the Romans, who named it Londinium.

The diagram illustrates coreference resolution for the word "London". Three instances of the word are highlighted with blue boxes: "London" at the start of the first sentence, "London" at the start of the second sentence, and "It" at the end of the second sentence. A blue curved line connects the first "London" box to the second "London" box. Another blue curved line connects the second "London" box to the "It" box, showing that all three refer to the same entity.

With coreference information combined with the parse tree and named entity information, we should be able to extract a lot of information out of this document!

Coreference resolution is one of the most difficult steps in our pipeline to implement. It’s even more difficult than sentence parsing. Recent advances in deep learning have resulted in new approaches that are more accurate, but it isn’t perfect yet.

Coding the NLP Pipeline in Python



Thanks to amazing python libraries like **spaCy**, it's already done! The steps are all coded and ready to use

We can install spaCy like this:

```
1 # Install spaCy
2 pip3 install -U spacy
3
4 # Download the large English model for spaCy
5 python3 -m spacy download en_core_web_lg
6
7 # Install textacy which will also be useful
8 pip3 install -U textacy
```


Then the code to run an NLP pipeline on a piece of text looks like this:

```
1  import spacy
2
3  # Load the large English NLP model
4  nlp = spacy.load('en_core_web_lg')
5
6  # The text we want to examine
7  text = """London is the capital and most populous city of England and
8  the United Kingdom. Standing on the River Thames in the south east
9  of the island of Great Britain, London has been a major settlement
10 for two millennia. It was founded by the Romans, who named it Londinium.
11 """
12
13 # Parse the text with spaCy. This runs the entire pipeline.
14 doc = nlp(text)
15
16 # 'doc' now contains a parsed version of text. We can use it to do anything we want!
17 # For example, this will print out all the named entities that were detected:
18 for entity in doc.ents:
19     print(f"{{entity.text}} ({{entity.label}})")
```

If we run that, we'll get a list of named entities and entity types detected in our document:

```
London (GPE)
England (GPE)
the United Kingdom (GPE)
the River Thames (FAC)
Great Britain (GPE)
London (GPE)
two millennia (DATE)
Romans (NORP)
Londinium (PERSON)
```

the meanings of all the entity codes are given in this link:

<https://spacy.io/usage/linguistic-features#entity-types>

Notice that it makes a mistake on “**Londinium**” and thinks it is the **name of a person** instead of a **place**. This is probably because there was nothing in the training data set similar to that and it made a best guess. **Named Entity Detection** often requires a little bit of model fine tuning if you are parsing text that has unique or specialized terms like this.

Suppose you've been given the task of removing any and all names from your documents. Here's a simple scrubber that removes all the names it detects:

```
1  import spacy
2
3  # Load the large English NLP model
4  nlp = spacy.load('en_core_web_lg')
5
6  # Replace a token with "REDACTED" if it is a name
7  def replace_name_with_placeholder(token):
8      if token.ent_iob != 0 and token.ent_type_ == "PERSON":
9          return "[REDACTED] "
10     else:
11         return token.string
12
13 # Loop through all the entities in a document and check if they are names
14 def scrub(text):
15     doc = nlp(text)
16     for ent in doc.ents:
17         ent.merge()
18     tokens = map(replace_name_with_placeholder, doc)
19     return "".join(tokens)
20
21 s = """
22 In 1950, Alan Turing published his famous article "Computing Machinery and Intelligence". In 1957,
23 Syntactic Structures revolutionized Linguistics with 'universal grammar', a rule based system of
24 """
25
26 print(scrub(s))
```

Extracting Facts

There's a python library called textacy that implements several common data extraction algorithms on top of spaCy. We can use it to search the parse tree for simple statements where the subject is "London" and the verb is a form of "be". That should help us find facts about London.

```
1  import spacy
2  import textacy.extract
3
4  # Load the large English NLP model
5  nlp = spacy.load('en_core_web_lg')
6
7  # The text we want to examine
8  text = """London is the capital and most populous city of England and the United Kingdom.
9  Standing on the River Thames in the south east of the island of Great Britain,
10 London has been a major settlement for two millennia. It was founded by the Romans,
11 who named it Londinium.
12 """
13
14 # Parse the document with spaCy
15 doc = nlp(text)
16
17 # Extract semi-structured statements
18 statements = textacy.extract.semistructured_statements(doc, "London")
19
20 # Print the results
21 print("Here are the things I know about London:")
22
23 for statement in statements:
24     subject, verb, fact = statement
25     print(f" - {fact}")
```

And here's what it prints:

```
Here are the things I know about London:
```

- the capital and most populous city of England and the United Kingdom.
- a major settlement for two millennia.

if you run that same code on the entire **London wikipedia** article text instead of just three sentences, you'll get a more impressive result: see next page

Here are the things I know about London:

- the capital and most populous city of England and the United Kingdom
- a major settlement for two millennia
- the world's most populous city from around 1831 to 1925
- beyond all comparison the largest town in England
- still very compact
- the world's largest city from about 1831 to 1925
- the seat of the Government of the United Kingdom
- vulnerable to flooding
- "one of the World's Greenest Cities" with more than 40 percent green space or open water
- the most populous city and metropolitan area of the European Union and the second most populous in Europe
- the 19th largest city and the 18th largest metropolitan region in the world
- Christian, and has a large number of churches, particularly in the City of London
- also home to sizeable Muslim, Hindu, Sikh, and Jewish communities
- also home to 42 Hindu temples
- the world's most expensive office market for the last three years according to world property journal (2015) report
- one of the pre-eminent financial centres of the world as the most important location for international finance
- the world top city destination as ranked by TripAdvisor users
- a major international air transport hub with the busiest city airspace in the world
- the centre of the National Rail network, with 70 percent of rail journeys starting or ending in London
- a major global centre of higher education teaching and research and has the largest concentration of higher education institutes in Europe
- home to designers Vivienne Westwood, Galliano, Stella McCartney, Manolo Blahnik, and Jimmy Choo, among others
- the setting for many works of literature
- a major centre for television production, with studios including BBC Television Centre, The Fountain Studios and The London Studios
- also a centre for urban music
- the "greenest city" in Europe with 35,000 acres of public parks, woodlands and gardens
- not the capital of England, as England does not have its own government

Going Beyond:

You will read the next topics on NLP over the next two days. Following are the links provided:

1. <https://medium.com/@ageitgey/text-classification-is-your-new-secret-weapon-7ca4fad15788>
2. <https://medium.com/@ageitgey/natural-language-processing-is-fun-part-3-explaining-model-predictions-486d8616813c>
3. <https://medium.com/@ageitgey/deepfaking-the-news-with-nlp-and-transformer-models-5e057ebd697d>
4. <https://medium.com/@ageitgey/build-your-own-google-translate-quality-machine-translation-system-d7dc274bd476>