Stanza – a Python library for processing many languages

Stanza (https://stanfordnlp.github.io/stanza/) is a Python library for natural language processing that provides pre-trained language models for almost 70 languages (https://stanfordnlp.github.io/stanza/available_models.html).

Stanza language models can perform tasks such as tokenization, part-of-speech tagging, morphological tagging and dependency parsing.

In [1]:

```
# Import the Stanza library
import stanza
```

To process a particular language, we must first download a Stanza language model using the `download()` . To download a language model for a given language, pass the two-letter language code (e.g. `fr` , `en` , `hi`) for the language as a string object to the `lang` argument.

For example, the following code would download a model for French.

```
# Download Stanza language model for French
stanza.download(lang='fr')
```

Stanza delivers models that have been trained on various datasets for some languages. Stanza refers to models that have been trained on various datasets as *packages*. By default, the package containing the model trained on the biggest dataset available for the language in question is downloaded.

To select a model trained on a specific dataset, pass the name of its package as a string object to the `package` argument.

The list of supported languages and their corresponding package names are provided in the list of language models (https://stanfordnlp.github.io/stanza/available_models.html) available for Stanza.

To install the language model into permanent storage, we must supply the model_dir parameter to the download() function, which contains a string pointing to a permanent storage location.

If the models are not saved permanently, they will be deleted when the server shuts down.

In [2]:

```
# Download a Stanza language model for French into the directory "../stanza_models"
stanza.download(lang='fr', model_dir='../stanza_models')
```

```
Downloading https://raw.githubusercontent.com/stanfordnlp/stanza-resource
s/main/resources_1.2.1.json: 139kB [00:00, 34.7MB/s]
2021-07-01 02:10:08 INFO: Downloading default packages for language: fr (F
rench)...
2021-07-01 02:10:09 INFO: File exists: ../stanza_models\fr\default.zip.
2021-07-01 02:10:17 INFO: Finished downloading models and saved to ../stan
za_models.
```

Loading a language model into Stanza

To load a Stanza language model into Python, we first create a *Pipeline* object by initialising an instance of the `Pipeline()` class from the `stanza` module.

Because we did **not** place the language model into the default directory, we also provide a string containing the path to the directory with Stanza language models to the `dir` argument.

We then store the pipeline under the variable `nlp_fr` . To load a language model for French into the pipeline, we must provide the string `fr` to the `lang` argument of the `Pipeline()` function.

In [3]:

```
# Initialize a Stanza pipeline with a language model for French, which is assigned to t
he variable 'nlp_fr' using the Pipeline() class.
nlp_fr = stanza.Pipeline(lang='fr', dir='../stanza_models')
```

```
2021-07-01 02:10:17 INFO: Loading these models for language: fr (French):
=======================
| Processor | Package |
-----------------------
| tokenize  | gsd     |
| mwt       | gsd     |
| pos       | gsd     |
| lemma     | gsd     |
| depparse  | gsd     |
| ner       | wikiner |
=======================

2021-07-01 02:10:17 INFO: Use device: cpu
2021-07-01 02:10:17 INFO: Loading: tokenize
2021-07-01 02:10:17 INFO: Loading: mwt
2021-07-01 02:10:17 INFO: Loading: pos
2021-07-01 02:10:18 INFO: Loading: lemma
2021-07-01 02:10:18 INFO: Loading: depparse
2021-07-01 02:10:19 INFO: Loading: ner
2021-07-01 02:10:22 INFO: Done loading processors!
```

Loading a language model into Stanza returns *Pipeline* object, which consists of a number of *processors* that perform various natural language processing tasks.

To speed up processing, we can define the processors to be included in the *Pipeline* object by providing the argument `processors` with a string object that contains the processor names (https://stanfordnlp.github.io/stanza/pipeline.html#processors) to be included in the pipeline.

For example, creating a *Pipeline* using the command below would only include the processors for tokenization and part-of-speech tagging into the pipeline.

```
# Initialise a Stanza pipeline with a language model for French;
# assign model to variable 'nlp_fr'.
# Only include tokenizer and part-of-speech tagger.
nlp_fr = stanza.Pipeline(lang='fr', processors='tokenize, pos')
```

Processing text using Stanza

Now we can feed some text in French to the models as a string object.

We store the result under the variable `doc_fr`.

In [4]:

```
# Feed French text to the model under 'nlp_fr'; store result under the variable 'doc_f
r'
doc_fr = nlp_fr("J'aime les chiens. J'aime aussi les chiens. Ce sont des animaux intell
igents.")
```

This creates a Stanza *Document* (https://stanfordnlp.github.io/stanza/data_objects.html#document) object, which contains the linguistic annotations created by passing the text through the pipeline.

The attribute `sentences` of a Stanza *Document* object contains a list, where each item contains a single sentence.

In [5]:

```
# Check the contents of `doc_fr`
print(doc_fr, sep="\n")
```

```
[
  [
    {
      "id": 1,
      "text": "J'",
      "lemma": "il",
      "upos": "PRON",
      "feats": "Number=Sing|Person=1|PronType=Prs",
      "head": 2,
      "deprel": "nsubj",
      "start_char": 0,
      "end_char": 2,
      "ner": "O"
    },
    {
      "id": 2,
      "text": "aime",
      "lemma": "aimer",
      "upos": "VERB",
      "feats": "Mood=Ind|Number=Sing|Person=1|Tense=Pres|VerbForm=Fin",
      "head": 0,
      "deprel": "root",
      "start_char": 2,
      "end_char": 6,
      "ner": "O"
    },
    {
      "id": 3,
      "text": "les",
      "lemma": "le",
      "upos": "DET",
      "feats": "Definite=Def|Number=Plur|PronType=Art",
      "head": 4,
      "deprel": "det",
      "start_char": 7,
      "end_char": 10,
      "ner": "O"
    },
    {
      "id": 4,
      "text": "chiens",
      "lemma": "chien",
      "upos": "NOUN",
      "feats": "Gender=Masc|Number=Plur",
      "head": 2,
      "deprel": "obj",
      "start_char": 11,
      "end_char": 17,
      "ner": "O"
    },
    {
      "id": 5,
      "text": ".",
      "lemma": ".",
      "upos": "PUNCT",
      "head": 2,
      "deprel": "punct",
      "start_char": 17,
      "end_char": 18,
      "ner": "O"
    }
    ],
    [
      {
        "id": 1,
        "text": "J'",
        "lemma": "il",
        "upos": "PRON",
        "feats": "Number=Sing|Person=1|PronType=Prs",
        "head": 2,
        "deprel": "nsubj",
        "start_char": 19,
        "end_char": 21,
        "ner": "O"
      },
      {
        "id": 2,
        "text": "aime",
        "lemma": "aimer",
        "upos": "VERB",
        "feats": "Mood=Ind|Number=Sing|Person=1|Tense=Pres|VerbForm=Fin",
        "head": 0,
        "deprel": "root",
        "start_char": 21,
        "end_char": 25,
        "ner": "O"
      },
      {
        "id": 3,
        "text": "aussi",
        "lemma": "aussi",
        "upos": "ADV",
        "head": 2,
        "deprel": "advmod",
        "start_char": 26,
        "end_char": 31,
        "ner": "O"
      },
      {
        "id": 4,
        "text": "les",
        "lemma": "le",
        "upos": "DET",
        "feats": "Definite=Def|Number=Plur|PronType=Art",
        "head": 5,
        "deprel": "det",
        "start_char": 32,
        "end_char": 35,
        "ner": "O"
      },
      {
        "id": 5,
        "text": "chiens",
        "lemma": "chien",
        "upos": "NOUN",
        "feats": "Gender=Masc|Number=Plur",
        "head": 2,
        "deprel": "obj",
        "start_char": 36,
        "end_char": 42,
        "ner": "O"
      },
    {
      "id": 6,
      "text": ".",
      "lemma": ".",
      "upos": "PUNCT",
      "head": 2,
      "deprel": "punct",
      "start_char": 42,
      "end_char": 43,
      "ner": "O"
    }
  ],
  [
    {
      "id": 1,
      "text": "Ce",
      "lemma": "ce",
      "upos": "PRON",
      "feats": "Gender=Masc|Person=3|PronType=Dem",
      "head": 4,
      "deprel": "nsubj",
      "start_char": 44,
      "end_char": 46,
      "ner": "O"
    },
    {
      "id": 2,
      "text": "sont",
      "lemma": "être",
      "upos": "AUX",
      "feats": "Mood=Ind|Number=Plur|Person=3|Tense=Pres|VerbForm=Fin",
      "head": 4,
      "deprel": "cop",
      "start_char": 47,
      "end_char": 51,
      "ner": "O"
    },
    {
      "id": 3,
      "text": "des",
      "lemma": "un",
      "upos": "DET",
      "feats": "Definite=Ind|Number=Plur|PronType=Art",
      "head": 4,
      "deprel": "det",
      "start_char": 52,
      "end_char": 55,
      "ner": "O"
    },
    {
      "id": 4,
      "text": "animaux",
      "lemma": "animal",
      "upos": "NOUN",
      "feats": "Gender=Masc|Number=Plur",
      "head": 0,
      "deprel": "root",
      "start_char": 56,
      "end_char": 63,
      "ner": "O"
    },
    {
      "id": 5,
      "text": "intelligents",
      "lemma": "intelligent",
      "upos": "ADJ",
      "feats": "Gender=Masc|Number=Plur",
      "head": 4,
      "deprel": "amod",
      "start_char": 64,
      "end_char": 76,
      "ner": "O"
    },
    {
      "id": 6,
      "text": ".",
      "lemma": ".",
      "upos": "PUNCT",
      "head": 4,
      "deprel": "punct",
      "start_char": 76,
      "end_char": 77,
      "ner": "O"
    }
  ]
]
```

Tokenization and Sentence Segmentation

In [6]:

```
nlp_fr = stanza.Pipeline(lang='fr', dir='../stanza_models', processors='tokenize')

doc_fr = nlp_fr("J'aime les chiens. J'aime aussi les chiens. Ce sont des animaux intell
igents.")

for i, sentence in enumerate(doc_fr.sentences):
    print(f'==========Sentence {i+1} tokens ==========')
    print(*[f'id: {token.id}\ttext: {token.text}' for token in sentence.tokens], sep="
\n")
```

```
2021-07-01 02:10:23 WARNING: Language fr package default expects mwt, whic
h has been added
2021-07-01 02:10:23 INFO: Loading these models for language: fr (French):
=======================
| Processor | Package |
-----------------------
| tokenize  | gsd     |
| mwt       | gsd     |
=======================

2021-07-01 02:10:23 INFO: Use device: cpu
2021-07-01 02:10:23 INFO: Loading: tokenize
2021-07-01 02:10:23 INFO: Loading: mwt
2021-07-01 02:10:23 INFO: Done loading processors!
==========Sentence 1 tokens ==========
id: (1,)        text: J'
id: (2,)        text: aime
id: (3,)        text: les
id: (4,)        text: chiens
id: (5,)        text: .
==========Sentence 2 tokens ==========
id: (1,)        text: J'
id: (2,)        text: aime
id: (3,)        text: aussi
id: (4,)        text: les
id: (5,)        text: chiens
id: (6,)        text: .
==========Sentence 3 tokens ==========
id: (1,)        text: Ce
id: (2,)        text: sont
id: (3,)        text: des
id: (4,)        text: animaux
id: (5,)        text: intelligents
id: (6,)        text: .
```

In [7]:

```
# To access segmented sentences
print([sentence.text for sentence in doc_fr.sentences])
```

```
["J'aime les chiens.", "J'aime aussi les chiens.", 'Ce sont des animaux in
telligents.']
```

Tokenization without Sentence Segmentation

In [8]:

```
nlp = stanza.Pipeline(lang='en', dir="../stanza_models", processors='tokenize', tokeniz
e_no_ssplit=True)
doc = nlp("J'aime les chiens.\n\nJ'aime aussi les chiens. Ce sont des animaux intellige
nts.")
for i, sentence in enumerate(doc.sentences):
    print(f'====== Sentence {i+1} tokens =======')
    print(*[f'id: {token.id}\ttext: {token.text}' for token in sentence.tokens], sep='
\n')
```

```
2021-07-01 02:10:24 INFO: Loading these models for language: en (English):
========================
| Processor | Package  |
------------------------
| tokenize  | combined |
========================

2021-07-01 02:10:24 INFO: Use device: cpu
2021-07-01 02:10:24 INFO: Loading: tokenize
2021-07-01 02:10:24 INFO: Done loading processors!
====== Sentence 1 tokens =======
id: (1,)        text: J'aime
id: (2,)        text: les
id: (3,)        text: chiens
id: (4,)        text: .
====== Sentence 2 tokens =======
id: (1,)        text: J'aime
id: (2,)        text: aussi
id: (3,)        text: les
id: (4,)        text: chiens
id: (5,)        text: .
id: (6,)        text: Ce
id: (7,)        text: sont
id: (8,)        text: des
id: (9,)        text: animaux
id: (10,)       text: intelligents
id: (11,)       text: .
```

In [9]:

```
nlp = stanza.Pipeline(lang='en', dir="../stanza_models", processors='tokenize', tokeniz
e_no_ssplit=False)
doc = nlp("J'aime les chiens.\n\nJ'aime aussi les chiens. Ce sont des animaux intellige
nts.")
for i, sentence in enumerate(doc.sentences):
    print(f'====== Sentence {i+1} tokens =======')
    print(*[f'id: {token.id}\ttext: {token.text}' for token in sentence.tokens], sep='
\n')
```

```
2021-07-01 02:10:24 INFO: Loading these models for language: en (English):
========================
| Processor | Package  |
------------------------
| tokenize  | combined |
========================

2021-07-01 02:10:24 INFO: Use device: cpu
2021-07-01 02:10:24 INFO: Loading: tokenize
2021-07-01 02:10:24 INFO: Done loading processors!
====== Sentence 1 tokens =======
id: (1,)        text: J'aime
id: (2,)        text: les
id: (3,)        text: chiens
id: (4,)        text: .
====== Sentence 2 tokens =======
id: (1,)        text: J'aime
id: (2,)        text: aussi
id: (3,)        text: les
id: (4,)        text: chiens
id: (5,)        text: .
====== Sentence 3 tokens =======
id: (1,)        text: Ce
id: (2,)        text: sont
id: (3,)        text: des
id: (4,)        text: animaux
id: (5,)        text: intelligents
id: (6,)        text: .
```

Accessing Part-of-Speech and Morphological Features

In [10]:

```
nlp_fr = stanza.Pipeline(lang='fr', dir='../stanza_models', processors='tokenize, pos')
doc_fr = nlp_fr('Nous avons atteint la fin du sentier.')
print(*[f'word: {word.text}\tupos: {word.upos}\txpos: {word.xpos}\tfeats: {word.feats i
f word.feats else "_"}' for sent in doc_fr.sentences for word in sent.words], sep='\n\n
')
```

```
2021-07-01 02:10:25 WARNING: Language fr package default expects mwt, whic
h has been added
2021-07-01 02:10:25 INFO: Loading these models for language: fr (French):
=======================
| Processor | Package |
-----------------------
| tokenize  | gsd     |
| mwt       | gsd     |
| pos       | gsd     |
=======================

2021-07-01 02:10:25 INFO: Use device: cpu
2021-07-01 02:10:25 INFO: Loading: tokenize
2021-07-01 02:10:25 INFO: Loading: mwt
2021-07-01 02:10:25 INFO: Loading: pos
2021-07-01 02:10:25 INFO: Done loading processors!
word: Nous      upos: PRON      xpos: None      feats: Number=Plur|Person=
1|PronType=Prs

word: avons     upos: AUX       xpos: None      feats: Mood=Ind|Number=Plu
r|Person=1|Tense=Pres|VerbForm=Fin

word: atteint   upos: VERB      xpos: None      feats: Gender=Masc|Number=
Sing|Tense=Past|VerbForm=Part

word: la        upos: DET       xpos: None      feats: Definite=Def|Gender
=Fem|Number=Sing|PronType=Art

word: fin       upos: NOUN      xpos: None      feats: Gender=Fem|Number=S
ing

word: de        upos: ADP       xpos: None      feats: _

word: le        upos: DET       xpos: None      feats: Definite=Def|Gender
=Masc|Number=Sing|PronType=Art

word: sentier   upos: NOUN      xpos: None      feats: Gender=Masc|Number=
Sing

word: . upos: PUNCT     xpos: None      feats: _
C:\Users\Administrator\.virtualenvs\Internship-py8mDiAO\lib\site-packages
\torch\_tensor.py:575: UserWarning: floor_divide is deprecated, and will b
e removed in a future version of pytorch. It currently rounds toward 0 (li
ke the 'trunc' function NOT 'floor'). This results in incorrect rounding f
or negative values.
To keep the current behavior, use torch.div(a, b, rounding_mode='trunc'),
or for actual floor division, use torch.div(a, b, rounding_mode='floor').
(Triggered internally at  ..\aten\src\ATen\native\BinaryOps.cpp:467.)
  return torch.floor_divide(self, other)
```

Accessing Lemma

In [11]:

```
nlp_fr = stanza.Pipeline(lang='fr', dir="../stanza_models", processors='tokenize, lemm
a')
doc_fr = nlp_fr('J\'aime les chiens. Ils sont intelligents.')
print(*[f'word: {word.text+" "}\tlemma: {word.lemma}' for sent in doc_fr.sentences for
word in sent.words], sep='\n')
```

```
2021-07-01 02:10:25 WARNING: Language fr package default expects mwt, whic
h has been added
2021-07-01 02:10:25 INFO: Loading these models for language: fr (French):
=======================
| Processor | Package |
-----------------------
| tokenize  | gsd     |
| mwt       | gsd     |
| lemma     | gsd     |
=======================

2021-07-01 02:10:25 INFO: Use device: cpu
2021-07-01 02:10:25 INFO: Loading: tokenize
2021-07-01 02:10:25 INFO: Loading: mwt
2021-07-01 02:10:25 INFO: Loading: lemma
2021-07-01 02:10:25 INFO: Done loading processors!
word: J'        lemma: il
word: aime      lemma: aimer
word: les       lemma: le
word: chiens    lemma: chien
word: .         lemma: .
word: Ils       lemma: il
word: sont      lemma: être
word: intelligents      lemma: intelligent
word: .         lemma: .
```

Accessing Syntactic Dependency Information

In [12]:

```
nlp_fr = stanza.Pipeline(lang='fr', dir='../stanza_models', processors='tokenize, pos,
 lemma, depparse')
doc_fr = nlp_fr('J\'aime les chiens. Ils sont intelligents.')
print(*[f'id: {word.id}\tword: {word.text}\thead id: {word.head}\thead: {sent.words[wor
d.head-1].text if word.head > 0 else "root"}\tdeprel: {word.deprel}' for sent in doc_fr
.sentences for word in sent.words], sep='\n')
```

```
2021-07-01 02:10:25 WARNING: Language fr package default expects mwt, whic
h has been added
2021-07-01 02:10:25 INFO: Loading these models for language: fr (French):
=======================
| Processor | Package |
-----------------------
| tokenize  | gsd     |
| mwt       | gsd     |
| pos       | gsd     |
| lemma     | gsd     |
| depparse  | gsd     |
=======================

2021-07-01 02:10:25 INFO: Use device: cpu
2021-07-01 02:10:25 INFO: Loading: tokenize
2021-07-01 02:10:25 INFO: Loading: mwt
2021-07-01 02:10:25 INFO: Loading: pos
2021-07-01 02:10:26 INFO: Loading: lemma
2021-07-01 02:10:26 INFO: Loading: depparse
2021-07-01 02:10:26 INFO: Done loading processors!
id: 1   word: J'        head id: 2      head: aime      deprel: nsubj
id: 2   word: aime      head id: 0      head: root      deprel: root
id: 3   word: les       head id: 4      head: chiens    deprel: det
id: 4   word: chiens    head id: 2      head: aime      deprel: obj
id: 5   word: . head id: 2      head: aime      deprel: punct
id: 1   word: Ils       head id: 3      head: intelligents      deprel: ns
ubj
id: 2   word: sont      head id: 3      head: intelligents      deprel: co
p
id: 3   word: intelligents      head id: 0      head: root      deprel: ro
ot
id: 4   word: . head id: 3      head: intelligents      deprel: punct
```

Accessing Named Entities for Sentence and Document

In [13]:

```
nlp_fr = stanza.Pipeline(lang='fr', dir='../stanza_models', processors='tokenize, ner')
doc_fr = nlp_fr("Chris Manning enseigne à l'Université de Stanford. Il vit dans la Bay
 Area.")

print(*[f'entity: {ent.text}\ttype: {ent.type}' for sent in doc_fr.sentences for ent in
sent.ents], sep='\n')
```

```
2021-07-01 02:10:26 WARNING: Language fr package default expects mwt, whic
h has been added
2021-07-01 02:10:26 INFO: Loading these models for language: fr (French):
=======================
| Processor | Package |
-----------------------
| tokenize  | gsd     |
| mwt       | gsd     |
| ner       | wikiner |
=======================

2021-07-01 02:10:26 INFO: Use device: cpu
2021-07-01 02:10:26 INFO: Loading: tokenize
2021-07-01 02:10:26 INFO: Loading: mwt
2021-07-01 02:10:26 INFO: Loading: ner
2021-07-01 02:10:29 INFO: Done loading processors!
entity: Chris Manning   type: PER
entity: Université de Stanford  type: ORG
entity: Bay Area        type: LOC
```

Sentiment Analysis

In [14]:

```
nlp_en = stanza.Pipeline(lang='en', dir='../stanza_models', processors='tokenize, senti
ment')

doc_en = nlp_en('I hate raw tomatoes. I like cats. I really love dogs.')
for i, sentence in enumerate(doc_en.sentences):
    print(i, sentence.sentiment)
```

```
2021-07-01 02:10:30 INFO: Loading these models for language: en (English):
========================
| Processor | Package  |
------------------------
| tokenize  | combined |
| sentiment | sstplus  |
========================

2021-07-01 02:10:30 INFO: Use device: cpu
2021-07-01 02:10:30 INFO: Loading: tokenize
2021-07-01 02:10:31 INFO: Loading: sentiment
2021-07-01 02:10:37 INFO: Done loading processors!
C:\Users\Administrator\.virtualenvs\Internship-py8mDiAO\lib\site-packages
\torch\nn\functional.py:652: UserWarning: Named tensors and all their asso
ciated APIs are an experimental feature and subject to change. Please do n
ot use them for anything important until they are released as stable. (Tri
ggered internally at  ..\c10/core/TensorImpl.h:1156.)
  return torch.max_pool1d(input, kernel_size, stride, padding, dilation, c
eil_mode)
0 0
1 1
2 2
```

Interfacing Stanza with spaCy

We can also use some of the Stanza language models in spaCy which can be achieved using a Python library named spacy-stanza (https://spacy.io/universe/project/spacy-stanza), which interfaces the two libraries.

Given that Stanza currently has more pre-trained language models available than spaCy, the spacy-stanza library considerably increases the number of language models available for spaCy.

There is, however, **one major limitation**: the language in question must be supported by both Stanza (https://stanfordnlp.github.io/stanza/available_models.html) and spaCy (https://spacy.io/usage/models#languages).

For example, we cannot use the Stanza language model for Hindi in spaCy, because spaCy does not support the Hindi language.

To start using Stanza language models in spaCy, let's start by importing the spacy-stanza library (module name: `spacy_stanza`).

In [15]:

```
# Import the spaCy and spacy-stanza libraries
import spacy
import spacy_stanza
```

This imports both spaCy and spacy-stanza libraries into Python. To continue, we must ensure that we have the Stanza language model for French available as well.

As shown above, this model can be downloaded using the following command:

```
# Download a Stanza language model for French
stanza.download(lang='fr')
```

Just as with the language model for Hindi above, we download the Stanza language model into the permanent storage on the CSC server.

To do so, provide a string object that points towards the directory `../stanza_models` to the argument `model_dir` of the `download()` function.

In [16]:

```
# Download a Stanza language model for French into the directory '../stanza_models'
stanza.download(lang='fr', model_dir='../stanza_models')
```

```
Downloading https://raw.githubusercontent.com/stanfordnlp/stanza-resource
s/main/resources_1.2.1.json: 139kB [00:00, 3.57MB/s]
2021-07-01 02:11:13 INFO: Downloading default packages for language: fr (F
rench)...
2021-07-01 02:11:20 INFO: File exists: ../stanza_models\fr\default.zip.
2021-07-01 02:11:30 INFO: Finished downloading models and saved to ../stan
za_models.
```

Because spaCy supports the French language (https://spacy.io/usage/models#languages), we can load Stanza language models for French into spaCy using the spacy-stanza library.

This can be achieved using the `load_pipeline()` function available under the `spacy_stanza` module.

To load Stanza language model for a given language, you must provide the two-letter code for the language in question (e.g. `fr`) to the argument `name` :

```
# Load a Stanza language model for French into spaCy
nlp_fr = spacy_stanza.load_pipeline(name='fr')
```

Because we did not download the Stanza language models into the default directory, we must also provide the optional argument `dir` to the `load_pipeline()` function.

The `dir` argument takes a string object as its input, which must point to the directory that contains Stanza language models.

In [17]:

```
# Use the load_pipeline function to load a Stanza model into spaCy.
# Assign the result under the variable 'nlp'.
nlp_fr = spacy_stanza.load_pipeline(name='fr', dir='../stanza_models')
```

```
2021-07-01 02:11:37 INFO: Loading these models for language: fr (French):
=======================
| Processor | Package |
-----------------------
| tokenize  | gsd     |
| mwt       | gsd     |
| pos       | gsd     |
| lemma     | gsd     |
| depparse  | gsd     |
| ner       | wikiner |
=======================

2021-07-01 02:11:37 INFO: Use device: cpu
2021-07-01 02:11:37 INFO: Loading: tokenize
2021-07-01 02:11:37 INFO: Loading: mwt
2021-07-01 02:11:37 INFO: Loading: pos
2021-07-01 02:11:37 INFO: Loading: lemma
2021-07-01 02:11:38 INFO: Loading: depparse
2021-07-01 02:11:38 INFO: Loading: ner
2021-07-01 02:11:39 INFO: Done loading processors!
```

If we examine the resulting object under the variable `nlp_fr` using Python's `type()` function, we will see that the object is indeed a spaCy *Language* object.

Generally, this object behaves just like any other spaCy *Language* object.

We feed the text as a string object to the *Language* object under `nlp_fr` and store the result under the variable `doc_fr` .

In [24]:

```
# Feed the text to the language model under 'nlp_fr', store result under 'doc_fr'
doc_fr = nlp_fr("J'aime les chiens. J'aime aussi les chiens. Ce sont des animaux intell
igents.")
```

Let's continue by retrieving sentences from the *Doc* object, which are available under the attribute `sents` .

The object available under the `sents` attribute is a Python generator that yields *Doc* objects.

To examine them, we must catch the objects into a suitable data structure. In this case, the data structure that best fits our needs is a Python list.

Hence we cast the output from the generator object under `sents` into a list using the `list()` function.

In [25]:

```
# Get sentences contained in the Doc object 'doc_fr'.
# Cast the result into list.
sents_fr = list(doc_fr.sents)

# Call the variable to check the output
sents_fr
```

Out[25]:

```
[J'aime les chiens.,
 J'aime aussi les chiens.,
 Ce sont des animaux intelligents.]
```

We can also use spaCy's `displacy` submodule to visualise the syntactic dependencies.

To do so for the first sentence under `sents_fr`, we must first access the first item in the list using brackets `[0]` as usual.

Let's start by checking the type of this object.

In [26]:

```
# Check the type of the first item in the list 'sents_fr'
type(sents_fr[0])
```

Out[26]:

```
spacy.tokens.span.Span
```

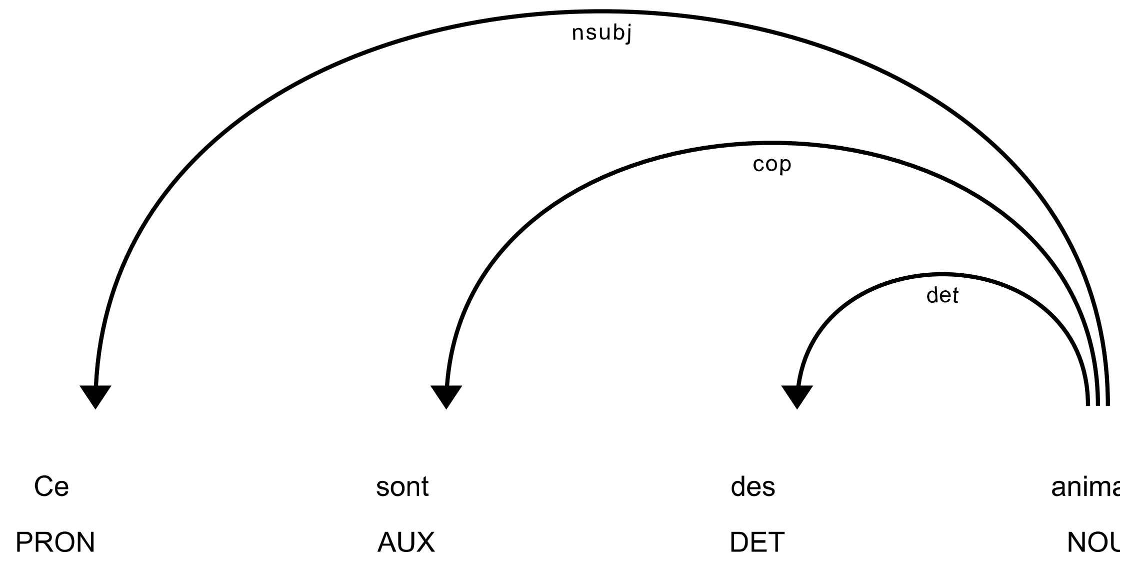
As you can see, the result is a spaCy *Span* object, which is a sequence of *Token* objects contained within a *Doc* object.

We can then call the `render` function from the `displacy` submodule to visualise the syntactic dependencies for the *Span* object under `sents_fr[0]`.

In [30]:

```python
# Import the display submodule
from spacy import displacy

# Use the render function to render the first item [0] in the list 'sents_fr'.
# Pass the argument 'style' with the value 'dep' to visualise syntactic dependencies.
displacy.render(sents_fr[2], style='dep')
```



Note that spaCy will raise a warning about storing custom attributes when writing the *Doc* object to disk for visualisation.

We can also examine the linguistic annotations created for individual *Token* objects within this *Span* object.

In [33]:

```
# Loop over each Token object in the Span
for token in sents_fr[0]:

    # Print the token, its lemma, dependency and morphological features
    print(token, token.lemma_, token.dep_, token.morph, "",  sep="\n")
```

```
J'
il
nsubj
Number=Sing|Person=1|PronType=Prs

aime
aimer
root
Mood=Ind|Number=Sing|Person=1|Tense=Pres|VerbForm=Fin

les
le
det
Definite=Def|Number=Plur|PronType=Art

chiens
chien
obj
Gender=Masc|Number=Plur

.
.
punct
```

The examples above show how we can access the linguistic annotations created by a Stanza language model through spaCy *Doc*, *Span* and *Token* objects.

In [ ]:

```

```