https://github.com/amrrs/custom-ner-with-spacy3/blob/main/Custom_NER_with_Spacy3.ipynb (https://github.com/amrrs/custom-ner-with-spacy3/blob/main/Custom_NER_with_Spacy3.ipynb)

Introduction to Named Entity Recognition

What is Named Entity Recognition?

Entities are words in a text that correspond to a specific type of data. They can be numerical, such as cardinal numbers;

temporal, such as dates; nominal, such as names of people and places; and political, such as geopolitical entities (GPE). In

short, an entity can be anything the designer wishes to designate as an item in a text that has a corresponding label.

Named entity recognition, or NER, is the process by which a system takes an input of unstructured data (a text) and outputs

structured data, specifically the identification of entities. Let us consider this short example.

Martha, a senior, moved to Spain where she will be playing basketball until 05 June 2022 or until she can't play any longer.

In this example, we have several potential entities. First, there is "Martha". Different NER models will have different

corresponding labels for such an entity, but PERSON or PER is considered standard practice. Note here that the label is

capitalized. This is also standard practice. We also have a GPE, or Geopolitical Entity, notably "Spain". Finally, we have a

DATE entity, "05 June 2022". These are standard labels that one can expect to extract from a text. If the domain at hand,

however, has additional labels, those can be extracted as well. Perhaps the client or user wants to not only extract people,

GPEs, and dates, but also sports. In such a scenario "basketball" could be extracted and given the label SPORT.

Not all entities are singular. As is common with texts, sometimes entities are Multi-word Tokens, or MWT. Let us consider the

same sentence as above, but with one modification:

Martha Thompson, a senior, moved to Spain where she will be playing basketball until 05 June 2022 or until she can't play any

longer.

Here, Martha now has a surname, "Thompson". We can either extract Martha and Thompson as individual entities or, as is more

common practice, extract both as a single entity, since "Martha Thompson" is a single individual. An NER system, therefore,

should recognize "Martha Thompson" as a single, MWT.

As we progress through these notebooks and videos, we will learn new NER concepts. For now, I recommend watching the video

below. Each notebook, including this one, will have a corresponding video lesson.

What are Frameworks?

In order to engage in NLP, a researcher must first decide upon the framework they wish to use. Framework is a word that

describes the software used by the researcher to engage in a specific task. A good way to think about a framework in Pythonic

terms is as a library, or packaged set of usable classes and functions to perform complex tasks easily. Deciding which framework

to use depends on a few variables. I will use the word "Pythonic" throughout this book. Pythonic is a term programmers of Python

use to refer to the standard, or community-accepted, way to do something. A good example is the way in which one imports pandas,

a library for analyzing and working with tabular data. When we import pandas, we import it as pd. Why? Because the documentation

told us to do so and, perhaps even more importantly, everyone in the community follows this syntax.

First, not all frameworks support all languages and not all frameworks support the same languages equally.

Second, certain frameworks perform certain tasks better than others. While all frameworks will tokenize equally well (usually),

the way in which some tasks, such as finding the root of words via lemmatization (spaCy) vs. stemming (Stanza) will vary.

Decision on a framework for this purpose typically lies in the realm of computational linguistics or distance reading for the

purpose of finding how a word (or words) appear in texts in all forms (conjugated and declined).

A common third thing to consider is the way in which the framework performs NLP. There are essentially two methods for

performing NLP: rules-based and machine learning-based. Rules-based NLP is the process by which the frameworks has a

predetermined set of rules for how to handle specific tasks. In order to find entities in a text, for example, a rules-based

method will contain a dictionary of all types of entities or it may contain a RegEx formula for identifying patterns that match

an entity.

Most frameworks today are moving away from a rules-based approach to NLP in favor of a machine learning-based approach. Machine

learning-based NLP is the process by which developers use statistics to teach a computer system (known as a model) to perform a

task based on past experiences (known as training). We will be speaking much more about machine learning-based NLP later in a

later notebook as spaCy, the chief subject of this notebook, is a machine learning-based Python library.

What is spaCy?

The spaCy (spelled correctly) library is a robust machine learning NLP library developed by Explosion AI, a Berlin based team of

computer scientists and computational linguists. It supports a wide variety of European languages out-of-the-box with

statistical models capable of parsing texts, identifying parts-of-speech, and extract entities. SpaCy is also capable of easily

improving or training from scratch custom models on domain-specific texts.

Sentence Tokenization

A common essential task of NLP is known as tokenization. We looked at tokenization briefly in the last notebook in which we

wanted to break a text into individual components. This is one form of tokenization known as word tokenization. There are,

however, many other forms, such as sentence tokenization. Sentence tokenization is precisely the same as word tokenization,

except instead of breaking a text up into individual word and punctuation components, we break a text up into individual

sentences.

If you are familiar with Python, you may be familiar with the built-in split() function which allows for a programmer to split a

text by whitespace (default) or by passing an argument of a string to define where to split a text, i.e. split(“.”). A common

practice (without NLP frameworks) is to split a text into sentences by simply using the split function, but this is ill-advised.

Let us consider the example below

Input

```
text = "Martin J. Thompson is known for his writing skills. He is also good at programming."

#Now, let's try and use the split function to split the text object based on punctuation.

new = text.split(".")

print (new)
```

Output

```
['Martin J', ' Thompson is known for his writing skills', ' He is also good at programming', '']
```

While we successfully were able to split the two sentences, we had the unfortunate result of splitting at Martin J. The reason

for this may be obvious. In English, it is common convention to indicate abbreviation with the same punctuation mark used to

indicate the end of a sentence. The reason for this extends to the early middle ages when Irish monks began to introduce

punctuation and spacing to better read Latin (a story for another day).

The very thing that makes texts easier to read, however, greatly hinders our ability to easily split sentences. For this reason,

another method is needed. This is where sentence tokenization comes into play. In order to see how sentence tokenization

differs, let's begin with our first spaCy usage.

Introduction to Training a Machine Learning Model in spaCy

In the last notebook, we created a basic training set for a machine learning model using spaCy's EntityRuler. We were able to do

this by making certain presumptions about things that are very likely or certainly going to fall under a specific label. Such an

approach to cultivating a training set is, by its nature, problematic. It will miss some entities and falsely label others. If

one wishes this to be the essential training set used to train a final model, I encourage a manual check. If, however, you want

to use this model as a baseline model that can be used to cultivate a better training set via Prodigy, then this method will

also work.

In this notebook, we will not be interested in the refining of this training set, rather the use of it to train a custom spaCy

machine learning NER model. The methods, therefore, will receive the chief focus on this notebook, not the results.

In 01.04: Machine Learning NER, we first met machine learning and some of the fundamentals of it. If you have not viewed that

notebook and the videos within, I encourage you to do so prior to working through this notebook as I will be assuming that you

have a basic understanding of machine learning.

The reason I prefer spaCy over other NLP frameworks is spaCy's ability to scale well (work on both small and big data) and it's

easy-to-use training process. An NER practitioner does not have to create a custom neural network via PyTorch/FastAI or

TensorFlow/Keras, all of which have a steep learning curve, despite being some of the easiest frameworks to use. Instead, users

of spaCy can take advantage of the predesigned CNN architecture behind the spaCy training process. In version 3.0 of spaCy

(nightly is available at the time of writing this notebook), due in early 2021, the user will also be able to customize this

neural network architecture, expanding spaCy's utility and customizability.

In order to take advantage of the spaCy training process, the user need only understand a few basic concepts, such as how the

data should look going into the training process (covered in the last notebook) and a few hyperperameters (the things we adjust

in the training process to try and find optimal results).

In [1]:
```
! pip install -U spacy -q # upgrading the spacy version
```

```
ERROR: Invalid requirement: '#'
```

In [2]:
```
! python -m spacy info
```

```
============================== Info about spaCy ==============================

spaCy version    3.3.1
Location         C:\Users\User\anaconda3\lib\site-packages\spacy
Platform         Windows-10-10.0.19044-SP0
Python version   3.9.12
Pipelines        en_core_web_sm (3.3.0)
```

In [3]:
```
import spacy
from spacy.tokens import DocBin
from tqdm import tqdm
nlp = spacy.blank("en")#load a new spacy model
db= DocBin()# create a docbin object
```

Preparing the Data

In [1]:
```
import json

with open('mednlp.json', 'r') as f:
    data = json.load(f)

print(data['examples'][0])
print(data)
```


```
e': 'flatulence', 'correct': None, 'human_annotations': [{'timestamp': '2020-
03-21T00:26:41.322000Z', 'annotator_id': 1, 'tagged_token_id': '8214556a-7584
-4d9b-86cd-5e09137ad904', 'name': 'Ashpat123', 'reason': 'exploration'}], 'mo
del_annotations': []}, {'id': '98968e14-6756-4174-9d3d-7abd58b3aa34', 'tag_i
d': 'c06bd022-6ded-44a5-8d90-f17685bb85a1', 'end': 833, 'start': 823, 'exampl
e_id': '18c2f619-f102-452f-ab81-d26f7e283ffe', 'tag_name': 'Medicine', 'valu
e': 'loperamide', 'correct': None, 'human_annotations': [{'timestamp': '2020-
03-21T00:26:12.759000Z', 'annotator_id': 1, 'tagged_token_id': '98968e14-6756
-4174-9d3d-7abd58b3aa34', 'name': 'Ashpat123', 'reason': 'exploration'}], 'mo
del_annotations': []}, {'id': 'a0a03c7b-cfad-41ee-9f5c-f8a802475994', 'tag_i
d': '03eb3e50-d4d8-4261-a60b-fa5aee5deb4a', 'end': 853, 'start': 852, 'exampl
e_id': '18c2f619-f102-452f-ab81-d26f7e283ffe', 'tag_name': 'MedicalConditio
n', 'value': ' ', 'correct': None, 'human_annotations': [], 'model_annotation
s': []}, {'id': 'bfbddfd4-38aa-43a7-9366-24c95829ac8c', 'tag_id': '03eb3e50-d
4d8-4261-a60b-fa5aee5deb4a', 'end': 469, 'start': 461, 'example_id': '18c2f61
9-f102-452f-ab81-d26f7e283ffe', 'tag_name': 'MedicalCondition', 'value': 'dia
rrhea', 'correct': None, 'human_annotations': [{'timestamp': '2020-03-21T00:2
5:06.921000Z', 'annotator_id': 1, 'tagged_token_id': 'bfbddfd4-38aa-43a7-9366
-24c95829ac8c', 'name': 'Ashpat123', 'reason': 'exploration'}], 'model_annota
tions': []}, {'id': 'd7d68c18-0d8e-4547-a2fa-81fdcaf3080e', 'tag_id': '03eb3e
```


Input data should be in the following format: TRAIN_DATA = [(TEXT AS A STRING, {“entities”: [(START, END, LABEL)]})]

Input code

import spacy

nlp = spacy.load("en_core_web_sm") text = "Treblinka is a small village in Poland. Wikipedia notes that Treblinka is not large." corpus = []

doc = nlp(text) for sent in doc.sents: corpus.append(sent.text)

nlp = spacy.blank("en")

ruler = nlp.add_pipe("entity_ruler")

patterns = [{"label": "GPE", "pattern": "Treblinka"}]

ruler.add_patterns(patterns)

TRAIN_DATA = [] for sentence in corpus: doc = nlp(sentence) entities = []

```
    for ent in doc.ents:
        entities.append([ent.start_char, ent.end_char, ent.label_])
    TRAIN_DATA.append([sentence, {"entities": entities}])

print (TRAIN_DATA)
```

output of the above code

[['Treblinka is a small village in Poland.', {'entities': [[0, 9, 'GPE']]}], ['Wikipedia notes that Treblinka is not large.', {'entities': [[21, 30, 'GPE']]}]]

In [5]:
```
training_data = {'classes' : ['MEDICINE', "MEDICALCONDITION", "PATHOGEN"], 'annot
for example in data['examples']:
  temp_dict = {}
  temp_dict['text'] = example['content']
  temp_dict['entities'] = []
  for annotation in example['annotations']:
    start = annotation['start']
    end = annotation['end']
    label = annotation['tag_name'].upper()
    temp_dict['entities'].append((start, end, label))
  training_data['annotations'].append(temp_dict)

print(training_data['annotations'][0])
```

```
{'text': "While bismuth compounds (Pepto-Bismol) decreased the number of bowel
movements in those with travelers' diarrhea, they do not decrease the length of
illness.[91] Anti-motility agents like loperamide are also effective at reducin
g the number of stools but not the duration of disease.[8] These agents should
be used only if bloody diarrhea is not present.[92]\n\nDiosmectite, a natural a
luminomagnesium silicate clay, is effective in alleviating symptoms of acute di
arrhea in children,[93] and also has some effects in chronic functional diarrhe
a, radiation-induced diarrhea, and chemotherapy-induced diarrhea.[45] Another a
bsorbent agent used for the treatment of mild diarrhea is kaopectate.\n\nRaceca
dotril an antisecretory medication may be used to treat diarrhea in children an
d adults.[86] It has better tolerability than loperamide, as it causes less con
stipation and flatulence.[94]", 'entities': [(360, 371, 'MEDICINE'), (383, 408,
'MEDICINE'), (104, 112, 'MEDICALCONDITION'), (679, 689, 'MEDICINE'), (6, 23, 'M
EDICINE'), (25, 37, 'MEDICINE'), (461, 470, 'MEDICALCONDITION'), (577, 589, 'ME
DICINE'), (853, 865, 'MEDICALCONDITION'), (188, 198, 'MEDICINE'), (754, 762, 'M
EDICALCONDITION'), (870, 880, 'MEDICALCONDITION'), (823, 833, 'MEDICINE'), (85
2, 853, 'MEDICALCONDITION'), (461, 469, 'MEDICALCONDITION'), (535, 543, 'MEDICA
LCONDITION'), (692, 704, 'MEDICINE'), (563, 571, 'MEDICALCONDITION')]}
```

How to Convert the Training Data to spaCy Binary Files

In a previous version of this textbook, we used spaCy 2. The way in which we train in spaCy 3 is entirely different. While it is

possible to work through some of the fundamentals in a script, I think it is best done in the terminal. Because this textbook is

a JupyterBook, which sits on top of many Jupyter Notebooks, we can execute terminal-based commands with an ! at the start of a

cell. In order to train a machine learning model, the first thing that we need to do is to create a spaCy binary object of that

training data. I find it is always good to use a function if a bit of code is going to repeated in the future, that way it can

be reproducible. The following function is available on spaCy's repo:

https://github.com/explosion/projects/blob/v3/pipelines/ner_demo/scripts/convert (https://github.com/explosion/projects/blob/v3/pipelines/ner_demo/scripts/convert). I modified it slightly to work better in this

textbook.

Let's break down what this function is doing. It's entire purpose is to convert our spaCy 2 formatted TRAIN_DATA into spaCy 3

binary training data. In my workflow, I like to keep these two steps separate. The reason is that I like to be able to examine

and verify my training data prior to converting it to spaCy 3 binary format. It allows for one final quick manual validation.

The function takes three arguments:

lang => this will be the language of the blank model. Use "en" for English, "de" for German, etc.

TRAIN_DATA => this will be the training data as a list of lists like we saw above.

output_path => this will be the output directory in which the spaCy binary file will sit.

The nice thing about this function is if your training data does not align, it will simply be ignored. This will prevent errors

in the training process. In order for this function to work, it needs to create a DocBin object to save. The db = DocBin()

allows us to run db.add() and add in our training data individually. These are then converted to binary objects (smaller in size

and faster to load). and saved to disk with db.to_disk().

Input code

```
import srsly
```

```
import typer
import warnings
from pathlib import Path
import spacy
from spacy.tokens import DocBin
def convert(lang: str, TRAIN_DATA, output_path: Path):
    nlp = spacy.blank(lang)
    db = DocBin()
    for text, annot in TRAIN_DATA:

        doc = nlp.make_doc(text)
        ents = []
        for start, end, label in annot["entities"]:
            span = doc.char_span(start, end, label=label)
            if span is None:
                msg = f"Skipping entity [{start}, {end}, {label}] in the fol
    lowing text because the character span '
{doc.text[start:end]}' does not align with token boundaries:\n\n{repr(text)}\n"
                warnings.warn(msg)
            else:
                ents.append(span)
        doc.ents = ents
        db.add(doc)
    db.to_disk(output_path)
```

In [6]:

```
from spacy.tokens import DocBin
from tqdm import tqdm

nlp = spacy.blank("en") # load a new spacy model
doc_bin = DocBin()# create a DocBin object
```

In [7]:

```
from spacy.util import filter_spans

for training_example  in tqdm(training_data['annotations']):
    text = training_example['text']
    labels = training_example['entities']
    doc = nlp.make_doc(text)
    ents = []
    for start, end, label in labels:
        span = doc.char_span(start, end, label=label, alignment_mode="contract")
        if span is None:
            print("Skipping entity")
        else:
            ents.append(span)
    filtered_ents = filter_spans(ents)
    doc.ents = filtered_ents
    doc_bin.add(doc)

doc_bin.to_disk("C:\\Users\\User\\training_data\\training_data.spacy ") # save th
```

```
  0%|
| 0/31 [00:00<?, ?it/s]

Skipping entity
Skipping entity
Skipping entity
Skipping entity
Skipping entity
Skipping entity
Skipping entity
Skipping entity
Skipping entity
Skipping entity
Skipping entity
Skipping entity
Skipping entity
Skipping entity
Skipping entity
Skipping entity
Skipping entity
Skipping entity

100%|
| 31/31 [00:00<00:00, 409.00it/s]


Skipping entity
Skipping entity
Skipping entity
Skipping entity
Skipping entity
Skipping entity
Skipping entity
Skipping entity
Skipping entity
Skipping entity
Skipping entity
```

In [8]:
```
import os
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"
```

What is the spaCy config.cfg File and How do I create it?

Now that we have our training data ready, it's time to start preparing our model. In spaCy 3, we have a lot of control over the

neural network architecture and hyperperameters of our model. This all takes place in the new config.cfg file. This config file

is giving to spaCy during the training process so that it knows what to train and how. In order to create the config.cfg file,

we first need to create a base_config.cfg file. To do that, we can use spaCy's handy GUI, found here:

https://spacy.io/usage/training (https://spacy.io/usage/training) (scroll down a bit). You will find something that looks like this:

For our purposes, select, "English", the language that we are training, "ner" only, the model we are training, "CPU" (GPU is a

bit more complex), and efficiency (quicker to train and smaller because there are no word vectors). You will copy and paste the

output in the GUI into your directory as "base_config.cfg". We will only make two minor changes to this base_config.cfg file. We

will specify the path of train and dev (seen under the first category of paths). We will set these to the location of our

train.spacy and valid.spacy files.

Now that the base_config file is setup correctly, it's time to convert it to a config.cfg file. To do that, we need to execute a

terminal command. Fortunately, we can do that here in Jupyter Notebook. I have placed my base_config file in the subfolder data.

By running the command below, spaCy reformats the base_config into a properly formatted config.cfg file.

In [9]:
```
! python -m spacy init config config.cfg --lang en --pipeline ner --optimize effi
```

```
[x] The provided output file already exists. To force overwriting the config
file, set the --force or -F flag.
```

Output for the above code

[+] Auto-filled config with all values

[+] Saved config

data\config.cfg

You can now add your data and train your pipeline:

python -m spacy train config.cfg --paths.train ./train.spacy --paths.dev ./dev.spacy

2021-08-10 08:02:33.694893: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library

cudart64_110.dll

With the config.cfg file in place, we can train our first model. In our case, I will be placing our models in the subfolder models/output. We run the command below, and we have a trained model.

. How to Train a spaCy 3 Model from the config.cfg File¶

In [10]:
```
! python -m spacy train config.cfg --output ./ --paths.train ./training_data/trai
```

```
[i] Saving to output directory: .
[i] Using CPU

=========================== Initializing pipeline ===========================
[+] Initialized pipeline

============================= Training pipeline =============================
[i] Pipeline: ['tok2vec', 'ner']
[i] Initial learn rate: 0.001
E    #       LOSS TOK2VEC  LOSS NER  ENTS_F  ENTS_P  ENTS_R  SCORE
---  ------  ------------  --------  ------  ------  ------  ------
  0       0          0.00    150.79    0.00    0.00    0.00    0.00
  7     200       2763.31   3219.19   77.11   78.24   76.02    0.77
 14     400        282.99    567.47   95.56   94.80   96.34    0.96
 22     600        204.75    285.75   97.98   97.58   98.37    0.98
 30     800        207.53    205.76   98.17   98.37   97.97    0.98
 40    1000        212.69    195.07   97.75   98.35   97.15    0.98
 52    1200        328.18    200.23   98.38   97.98   98.78    0.98
 65    1400        194.97    213.20   98.37   98.77   97.97    0.98
 82    1600       2678.38    236.38   98.38   97.98   98.78    0.98
103    1800        135.31    207.07   98.37   98.37   98.37    0.98
128    2000       6217.45    308.34   97.97   97.97   97.97    0.98
160    2200        174.58    286.08   98.57   98.78   98.37    0.99
200    2400        211.39    271.12   98.78   99.18   98.37    0.99
249    2600        157.07    277.65   98.78   98.78   98.78    0.99
298    2800         95.16    265.10   98.78   99.18   98.37    0.99
348    3000         88.60    276.55   98.78   98.78   98.78    0.99
397    3200        121.22    261.26   98.79   98.39   99.19    0.99
447    3400        159.34    264.71   98.79   98.39   99.19    0.99
496    3600        139.03    262.88   98.78   98.78   98.78    0.99

[2022-07-10 13:44:23,334] [INFO] Set up nlp object from config
[2022-07-10 13:44:23,344] [INFO] Pipeline: ['tok2vec', 'ner']
[2022-07-10 13:44:23,347] [INFO] Created vocabulary
[2022-07-10 13:44:23,350] [INFO] Finished initializing nlp object
[2022-07-10 13:44:23,677] [INFO] Initialized pipeline components: ['tok2vec',
'ner']

545    3800        230.04    272.64   98.79   98.39   99.19    0.99
595    4000        100.13    260.13   98.78   98.78   98.78    0.99
644    4200         90.48    253.19   98.79   98.39   99.19    0.99
693    4400         74.63    257.99   98.78   98.78   98.78    0.99
742    4600        177.08    250.83   98.78   98.78   98.78    0.99
791    4800         82.74    256.17   98.79   98.39   99.19    0.99
[+] Saved pipeline to output directory
model-last
```

The above output tells us the epochs, the number of samples, as well as some metrics for our model. Our model shows 100%, but

this does not mean we have a good model. It is overfitted, meaning it essentially memorized one sample, Treblinka. Nevertheless,

let's load up the model and see how it performs.

In [11]:
```
nlp_ner = spacy.load("model-best")
```

Note that we gave the machine learning model NER a new sentence and it correctly identifies Treblinka as a "GPE". But we should

not get too excited. Minor alterations to this text result in a missed entity.

Input Code

trained_nlp = spacy.load("models/output/model-best")

text = "The village of Treblinka is located in Poland."

doc = trained_nlp(text)

for ent in doc.ents:

    print (ent.text, ent.label_)

out put

Treblinka GPE

In [12]:
```
doc = nlp_ner('''1)Patient presented with complains of SOB, fast, rapid & shallow
cough with phlegm, distress, bluish discolouration, muscle weakness, fever, chest
 In setting of pulmonary manifestation, lifestyle habits (alcohol, drugs, smoking
(flu, SARS-COVID, pneumonia, sepsis), smoke inhalation, trauma, inflammatory cond
blood transfusion, fat embolism, severe burns, injury, family history. Physical e
cough, rales, crackles.

2)A 65-year-old female with past medical history of aortic stenosis, HTN, obesity

3)A 42-year-old male with past medical history of HTN, presented to ED after havi
Patient presented with syncope, SOB, dizziness. During the stay, physical examina

4)This is a 13-year-old female with PMHx significant for autism, ADHD and moderat

5)Type 2 diabetes mellitus with diabetic peripheral angiopathy with gangrene/ Gas
The patient presented with right foot ulcer with infection. Patient reports that

6)On 9/27/21 the preop/postop diagnosis the abscess gas forming organism right ca
```

text = "Mark, from New York, said that he wants to go to Treblinkaa to speak to the locals."

```
doc = trained_nlp(text)

for ent in doc.ents:

    print (ent.text, ent.label_)

if len(doc.ents) == 0:

    print ("No entities found.")
```

Output

No entities found.

In [13]:
```
from spacy import displacy
displacy.render(doc, style="ent", jupyter=True)
```

1)Patient presented with complains of SOB, fast **MEDICALCONDITION** , rapid & shallow breathing, rapid heart rate **MEDICALCONDITION** ,

cough with phlegm, distress, bluish discolouration, muscle weakness, fever, chest pain **MEDICALCONDITION** , fatigue, confusion, AMS.

In setting of pulmonary manifestation, lifestyle habits (alcohol, drugs, smoking), infection

(flu, SARS-COVID, pneumonia **MEDICALCONDITION** , sepsis), smoke inhalation, trauma,

inflammatory conditions(pancreatitis),

blood transfusion, fat embolism, severe burns, injury, family history. Physical examination was

evident for Dyspnea **MEDICINE** , hypoxemia, tachycardia, tachypnoea **MEDICALCONDITION**

hypotension **MEDICALCONDITION** cyanosis **MEDICALCONDITION**

Why does our model now fail?

Because we have trained a machine learning model, not an EntityRuler. It knows that Treblinka is a

GPE, but it has only learned to identify it if it is spelled correctly. This is a bad model. Machine learing NER models improve

with the more training data that we feed them. Most importantly, however, they improve with the greater amount of varied

training data we feed them. A good rule of thumb is to start with 200 training samples and then make adjustments going forward.

You may need to gather more varied training data or you may need to reconsider your labels. Another possibility is that you need

to fine-tune your hyperperameters in the config.cfg file. We will be covering these problems and solutions throughout the

remainder of this textbook. By now, though, you should have a good sense of how the training process works in spaCy 3. The

material discussed in this notebook are by far the most challenging so far. Take your time here and get to know this process

well before moving forward.

In [14]:
```
doc1 = nlp_ner('''Allowing Corbevax as booster for those vaccinated with Covishie

''')
spacy.displacy.render(doc1, style="ent", jupyter=True)
```

Allowing Corbevax **PATHOGEN** as booster for those vaccinated with Covishield, Covaxin likely

to be considered by NTAGI today

In [ ]:
```

```