NER Dataset builder

On va importer le fichier .csv qui va contenir les phrases d'entrainement de notre model qui ont été récoltés auprès de nos utilisateurs. C'est notre base à partir de laquelle on va pouvoir entrainer notre bot.

```
# Import training phrases csv file
import csv

csv_path = "/content/corpus.csv"

corpus = []

with open(csv_path, 'r') as f:
    reader = csv.reader(f)
    for row in reader:
        corpus.append({'text': row[0], 'dep': row[1],
                        'des': row[2]})

corpus.pop(0)

print(corpus)
```

```
[{'text': 'Je cherche un train depuis Paris pour Marseille', 'dep': 'Paris',
```

```
# Get sample of the corpus
print(corpus[0])
```

```
text': 'Je cherche un train depuis Paris pour Marseille', 'dep': 'Paris', 'des
```

Training Sets

Les données pour entrainées un model de machine learning existent sous trois formes:

- Les données d'entrainement
- Les données de validation
- Les données de test

Toutes ces données vont être strucutrées de la même manière, c'est à dire sous la forme d'une liste de données structurées dans laquelle chaque index va contenir une chaine de caractère qui va représenter un texte (la requête de l'utilisateur).

Le second composant de nos données d'entrainement va va être composé d'une liste d'entités à extraire présentes dans ce texte avec pour chacune d'elle la position de départ dans la string, la position de fin, ainsi que le label auquel cette entité doit être assosiées.

Durant la phase d'entrainement de notre model, ces annotations vont permettre au réseau de neurone (architecture de ML utilisé par spaCy) d'apprendre de ces données pour être capable d'identifier correctement les entitées sur lequelles on souhaite l'entrainer.

Voici la forme que doit prendre notre set d'entrainement:

```
TRAIN_DATA = [ (TEXT AS A STRING, {“entities”: [(START, END, LABEL)]}) ]
```

```
import re

TRAIN_DATA = []

# Get start and end position of the token inside the text value
def get_position(text, token):
  for match in re.finditer(token, text):
    return (match.start(), match.end())

# Get entity format (START, END, LABEL)
def get_entity(text, token, label):
  start, end = get_position(text, token)
  return (start, end, label)

for sentence in corpus:
  text = sentence["text"]
  get_entity(text, sentence["des"], "DES")
  get_entity(text, sentence["dep"], "DEP")
  # item => (TEXT AS A STRING, {“entities”: [(START, END, LABEL)]})
  item = (text, {"entities": [get_entity(text, sentence["dep"], "DEP"), get_entity
  TRAIN_DATA.append(item)

print(TRAIN_DATA)
```

```
[('Je cherche un train depuis Paris pour Marseille', {'entities': [(27, 32, '
```

```
# Get sample of training set
print(TRAIN_DATA[0])
```

```
('Je cherche un train depuis Paris pour Marseille', {'entities': [(27, 32, 'D
```

Conception du model NER

```
from pathlib import Path

model = None
output_dir=Path("/content")
n_iter=100
```

```
import spacy
```

```
#load the model
if model is not None:
    nlp = spacy.load(model)
    print("Loaded model '%s'" % model)
else:
    nlp = spacy.blank('fr')
    print("Created blank 'fr' model")

#set up the pipeline

# create the built-in pipeline components and add them to the pipeline

# nlp.create_pipe works for built-ins that are registered with spaCy
if "ner" not in nlp.pipe_names:
  ner = nlp.create_pipe("ner")
  nlp.add_pipe(ner, last=True)
# otherwise, get it so we can add labels
else:
  ner = nlp.get_pipe("ner")

# add labels
for _, annotations in TRAIN_DATA:
  for ent in annotations.get("entities"):
    ner.add_label(ent[2])

# if 'ner' not in nlp.pipe_names:
#     ner = nlp.create_pipe('ner')
#     nlp.add_pipe(ner, last=True)
# else:
#     ner = nlp.get_pipe('ner')
```

```
     Created blank 'fr' model
```

```
from spacy.util import minibatch, compounding

# get names of other pipes to disable them during training
other_pipes = [pipe for pipe in nlp.pipe_names if pipe != "ner"]
with nlp.disable_pipes(*other_pipes):  # only train NER
  # reset and initialize the weights randomly — but only if we're
  # training a new model
  if model is None:
    nlp.begin_training()
    for itn in range(n_iter):
      random.shuffle(TRAIN_DATA)
      losses = {}
      # batch up the examples using spaCy's minibatch
      batches = minibatch(TRAIN_DATA, size=compounding(4.0, 32.0, 1.001))
      for batch in batches:
        texts, annotations = zip(*batch)
        nlp.update(
          texts,  # batch of texts
          annotations,  # batch of annotations
          drop=0.5,  # dropout - make it harder to memorise data
          losses=losses)
      print("Losses", losses)
```

```
Losses {'ner': 1.3916099143419973}
Losses {'ner': 1.3916099143428506}
Losses {'ner': 0.026319758676895306}
Losses {'ner': 0.026629229643507876}
Losses {'ner': 0.026630696678555024}
Losses {'ner': 0.02663069685089419}
Losses {'ner': 0.026630698083238896}
Losses {'ner': 0.026630728479938644}
Losses {'ner': 0.026630728479940073}
Losses {'ner': 5.6550710868805856e-05}
Losses {'ner': 5.66405231803134e-05}
Losses {'ner': 5.7883449991872474e-05}
Losses {'ner': 5.78839667151033e-05}
Losses {'ner': 0.00019338895552494642}
Losses {'ner': 0.0001954374556215525}
Losses {'ner': 0.00019543755365130603}
Losses {'ner': 2.1960554911881758e-05}
Losses {'ner': 2.1974068558850085e-05}
Losses {'ner': 2.277183966689847e-05}
Losses {'ner': 2.390051657303369e-05}
Losses {'ner': 2.390217100135504e-05}
Losses {'ner': 0.058865171646796985}
Losses {'ner': 0.0588651716467978}
Losses {'ner': 4.6006127800468336e-08}
Losses {'ner': 3.871718260959478e-07}
Losses {'ner': 0.000518137245010829}
Losses {'ner': 0.017713386221446643}
Losses {'ner': 0.017713388932562517}
Losses {'ner': 0.01771339008664138}
Losses {'ner': 0.01771339008664138}
Losses {'ner': 9.785097321464183e-13}
Losses {'ner': 4.624630389596829e-09}
Losses {'ner': 0.7120139419118031}
Losses {'ner': 2.5234448241065834}
Losses {'ner': 2.523444824125201}
Losses {'ner': 2.5234448244979504}
Losses {'ner': 2.523444824508708}
Losses {'ner': 7.275113649981011e-11}
Losses {'ner': 1.5933890762885126e-05}
Losses {'ner': 1.593533758557636e-05}
Losses {'ner': 1.5935688954928646e-05}
Losses {'ner': 0.021060088056336156}
Losses {'ner': 0.021060095931450855}
Losses {'ner': 0.021060095931451892}
Losses {'ner': 8.422834691309759e-08}
Losses {'ner': 9.623176411315118e-07}
Losses {'ner': 9.707071704755867e-06}
Losses {'ner': 1.7610444341768459}
Losses {'ner': 1.7610444376032652}
Losses {'ner': 1.761044437615455}
Losses {'ner': 1.761044440491387}
Losses {'ner': 6.979546433063394e-08}
Losses {'ner': 0.10656479124684105}
Losses {'ner': 0.24224956457319663}
Losses {'ner': 0.24224984544007105}
Losses {'ner': 0.2422522389866073}
Losses {'ner': 0.24225223902603077}
Losses {'ner': 0.24225223903433227}
```

```
    Losses {'ner': 0.2422322390343322 7}
```

Tester le model

```
 # test the trained model

 # IOB code of named entity tag. 3 means the token begins an entity, 2 means it
 # is outside an entity, 1 means it is inside an entity, and 0 means no entity
 # tag is set.

for text, _ in TRAIN_DATA:
  doc = nlp(text)
  print("Entities", [(ent.text, ent.label_) for ent in doc.ents])
  print("Tokens", [(t.text, t.ent_type_, t.ent_iob) for t in doc])
```

```
    Entities [('Lyon', 'DES'), ('Saint-Etienne', 'DEP')]
    Tokens [('Un', '', 2), ('train', '', 2), ('pour', '', 2), ('Lyon', 'DES', 3),
    Entities [('Nice', 'DEP'), ('Lyon', 'DES')]
    Tokens [('Je', '', 2), ('recherche', '', 2), ('un', '', 2), ('train', '', 2),
    Entities [('Marseille', 'DES'), ('Grenoble', 'DEP')]
    Tokens [('J’', '', 2), ('ai', '', 2), ('besoin', '', 2), ('de', '', 2), ('me'
    Entities [('Paris', 'DEP'), ('Lyon', 'DES')]
    Tokens [('Je', '', 2), ('suis', '', 2), ('à', '', 2), ('la', '', 2), ('recher
    Entities [('Rennes', 'DEP'), ('Bordeaux', 'DES')]
    Tokens [('Un', '', 2), ('train', '', 2), ('de', '', 2), ('Rennes', 'DEP', 3),
    Entities [('Orléans', 'DEP'), ('Rouen', 'DES')]
    Tokens [('Donne', '', 2), ('moi', '', 2), ('un', '', 2), ('train', '', 2), ('
    Entities [('Lille', 'DES'), ('Gare du Nord', 'DEP')]
    Tokens [('Je', '', 2), ('veux', '', 2), ('partir', '', 2), ('à', '', 2), ('Li
    Entities [('Troyes', 'DEP'), ('Reims', 'DES')]
    Tokens [('De', '', 2), ('Troyes', 'DEP', 3), ('à', '', 2), ('Reims', 'DES', 3
    Entities [('Paris', 'DES'), ('Strasbourg', 'DEP')]
    Tokens [('Est', '', 2), ('ce', '', 2), ('qu’', '', 2), ('il', '', 2), ('y’',
    Entities [('Bordeaux', 'DEP'), ('Biarritz', 'DES')]
    Tokens [('Y’', '', 2), ('a', '', 2), ('-', '', 2), ('t', '', 2), ('-', '', 2)
    Entities [('Nice', 'DEP'), ('Fréjus', 'DES')]
    Tokens [('Trouve', '', 2), ('moi', '', 2), ('un', '', 2), ('train', '', 2), (
    Entities [('Reims', 'DEP'), ('Paris', 'DES')]
    Tokens [('Peux', '', 2), ('tu', '', 2), ('me', '', 2), ('trouver', '', 2), ('
    Entities [('Paris', 'DES'), ('Nancy', 'DEP')]
    Tokens [('Comment', '', 2), ('je', '', 2), ('me', '', 2), ('rend', '', 2), ('
    Entities [('Clermont-Ferrand', 'DES'), ('Vichy', 'DEP')]
    Tokens [('Je', '', 2), ('cherche', '', 2), ('un', '', 2), ('train', '', 2), (
    Entities [('Paris', 'DEP'), ('Reims', 'DES')]
    Tokens [('Un', '', 2), ('trajet', '', 2), ('de', '', 2), ('Paris', 'DEP', 3),
    Entities [('Reims', 'DEP'), ('Châlons-en-Champagne', 'DES')]
    Tokens [('Y’', '', 2), ('a', '', 2), ('-', '', 2), ('t', '', 2), ('-', '', 2)
    Entities [('Lyon', 'DES'), ('Nîmes', 'DEP')]
    Tokens [('Un', '', 2), ('trajet', '', 2), ('pour', '', 2), ('Lyon', 'DES', 3)
    Entities [('Paris', 'DEP'), ('Lyon', 'DES')]
    Tokens [('Comment', '', 2), ('je', '', 2), ('peux', '', 2), ('aller', '', 2),
    Entities [('Gare de l’Est', 'DEP'), ('Strasbourg', 'DES')]
    Tokens [('Est', '', 2), ('ce', '', 2), ('qu’', '', 2), ('il', '', 2), ('y’',
    Entities [('Paris', 'DEP'), ('Marseille', 'DES')]
    Tokens [('Je', '', 2), ('cherche', '', 2), ('un', '', 2), ('train', '', 2), (
    Entities [('Chambéry', 'DEP'), ('Gare de Lyon', 'DES')]
```

```
 Tokens [('Obtient', '', 2), ('moi', '', 2), ('un', '', 2), ('train', '', 2),
 Entities [('Reims', 'DES'), ('Strasbourg', 'DEP')]
 Tokens [('Un', '', 2), ('train', '', 2), ('pour', '', 2), ('Reims', 'DES', 3)
 Entities [('Saint-Quentin', 'DES'), ('Meaux', 'DEP')]
 Tokens [('Quel', '', 2), ('est', '', 2), ('le', '', 2), ('meilleur', '', 2),
 Entities [('Paris', 'DEP'), ('Hyères', 'DES')]
 Tokens [('Je', '', 2), ('veux', '', 2), ('obtenir', '', 2), ('le', '', 2), ('
 Entities [('Paris', 'DEP'), ('Valence', 'DES')]
 Tokens [('Trouve', '', 2), ('moi', '', 2), ('un', '', 2), ('trajet', '', 2),
```

```
for text, _ in TRAIN_DATA:
    doc = nlp(text)
    print('Entities', [(ent.text, ent.label_) for ent in doc.ents])
```

```
 Entities [('Lyon', 'DES'), ('Saint-Etienne', 'DEP')]
 Entities [('Nice', 'DEP'), ('Lyon', 'DES')]
 Entities [('Marseille', 'DES'), ('Grenoble', 'DEP')]
 Entities [('Paris', 'DEP'), ('Lyon', 'DES')]
 Entities [('Rennes', 'DEP'), ('Bordeaux', 'DES')]
 Entities [('Orléans', 'DEP'), ('Rouen', 'DES')]
 Entities [('Lille', 'DES'), ('Gare du Nord', 'DEP')]
 Entities [('Troyes', 'DEP'), ('Reims', 'DES')]
 Entities [('Paris', 'DES'), ('Strasbourg', 'DEP')]
 Entities [('Bordeaux', 'DEP'), ('Biarritz', 'DES')]
 Entities [('Nice', 'DEP'), ('Fréjus', 'DES')]
 Entities [('Reims', 'DEP'), ('Paris', 'DES')]
 Entities [('Paris', 'DES'), ('Nancy', 'DEP')]
 Entities [('Clermont-Ferrand', 'DES'), ('Vichy', 'DEP')]
 Entities [('Paris', 'DEP'), ('Reims', 'DES')]
 Entities [('Reims', 'DEP'), ('Châlons-en-Champagne', 'DES')]
 Entities [('Lyon', 'DES'), ('Nîmes', 'DEP')]
 Entities [('Paris', 'DEP'), ('Lyon', 'DES')]
 Entities [('Gare de l’Est', 'DEP'), ('Strasbourg', 'DES')]
 Entities [('Paris', 'DEP'), ('Marseille', 'DES')]
 Entities [('Chambéry', 'DEP'), ('Gare de Lyon', 'DES')]
 Entities [('Reims', 'DES'), ('Strasbourg', 'DEP')]
 Entities [('Saint-Quentin', 'DES'), ('Meaux', 'DEP')]
 Entities [('Paris', 'DEP'), ('Hyères', 'DES')]
 Entities [('Paris', 'DEP'), ('Valence', 'DES')]
```

✓ 0 s terminée à 19:10 ● ✕