*Information Extraction Document*

1. **Project Environnement**
2. **Framework or platform of choice :**

Colab



The decision to employ Google Colab for our NER information extraction project was driven by the imperative for GPU acceleration. Colab's provision of free GPU resources empowers us to tackle heavy NER models efficiently, ultimately resulting in faster model development and more expedited research progress. This choice aligns perfectly with our objective of extracting high-quality named entities from text with maximum efficiency.

By harnessing the GPU capabilities offered by Google Colab, we are better equipped to address the challenges posed by NER tasks and advance our research in the field of information extraction.

1. **Libraries used in the project :**
2. **Spacy :**



**This is how we will utilize spacy in our project =>**

* Defininig Custom Categories: Tailor SpaCy's NER to recognize job-specific entities like JOB\_TITLE, SKILL, and LOCATION and the other attributes we categorized, which are essential for extracting relevant information.
* Train a Custom NER Model: Train SpaCy's NER with annotated job postings data, enabling the model to learn patterns and context for accurate entity recognition.
* Fine-Tune Pre-trained Models: Enhance SpaCy's pre-trained models with domain-specific entities, improving their accuracy in identifying IT skills, company names etc

1. **Python** ( version 3.10 ) :

Python was selected as the primary programming language for our Named Entity Recognition (NER) information extraction project due to Python's versatility, extensive NLP support, ease of use, and compatibility with machine learning libraries make it the ideal choice for our NER information extraction project

1. **Transformers**:

**Transformers** : Well-suited for capturing contextual information and extracting attributes from different parts of job ads, captures long-range dependencies, sensitive to hyperparameter tuning, state-of-the-art performance in NLP tasks

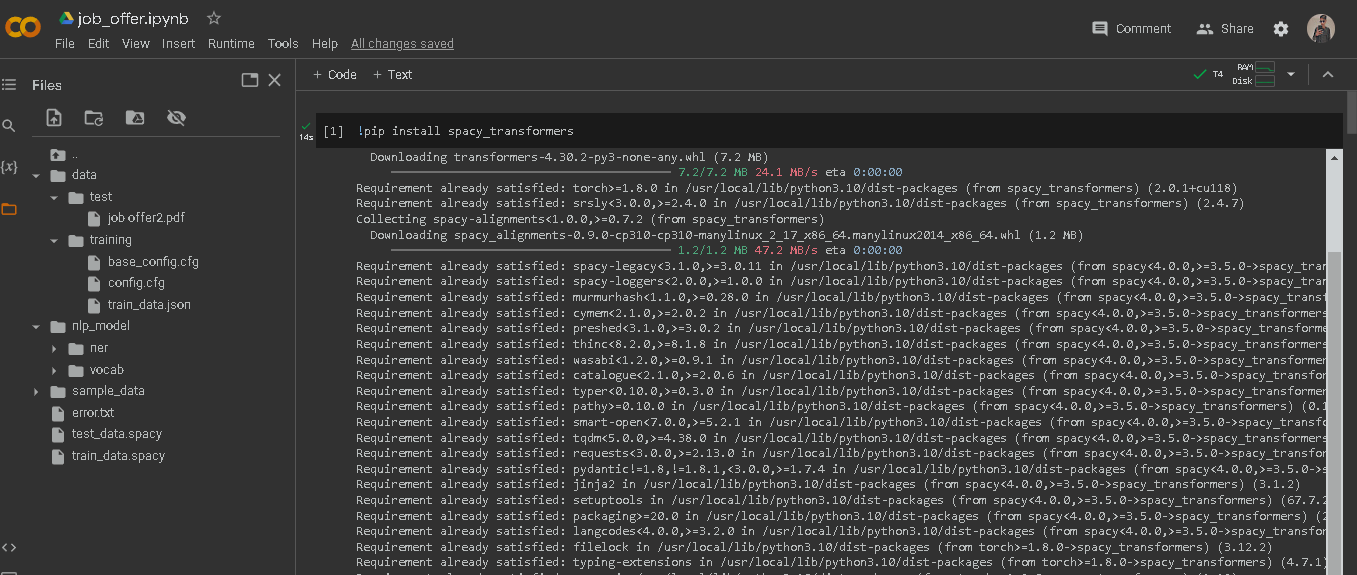
**Implementation:** Utilize a pre-trained transformer model, in our case : RoBERTa, with a sequence tagging or classification head. Fine-tune the model using a labeled dataset of job advertisements

**Output:** The model will output predicted labels or spans for various elements in the job advertisements

==> GPU recommended

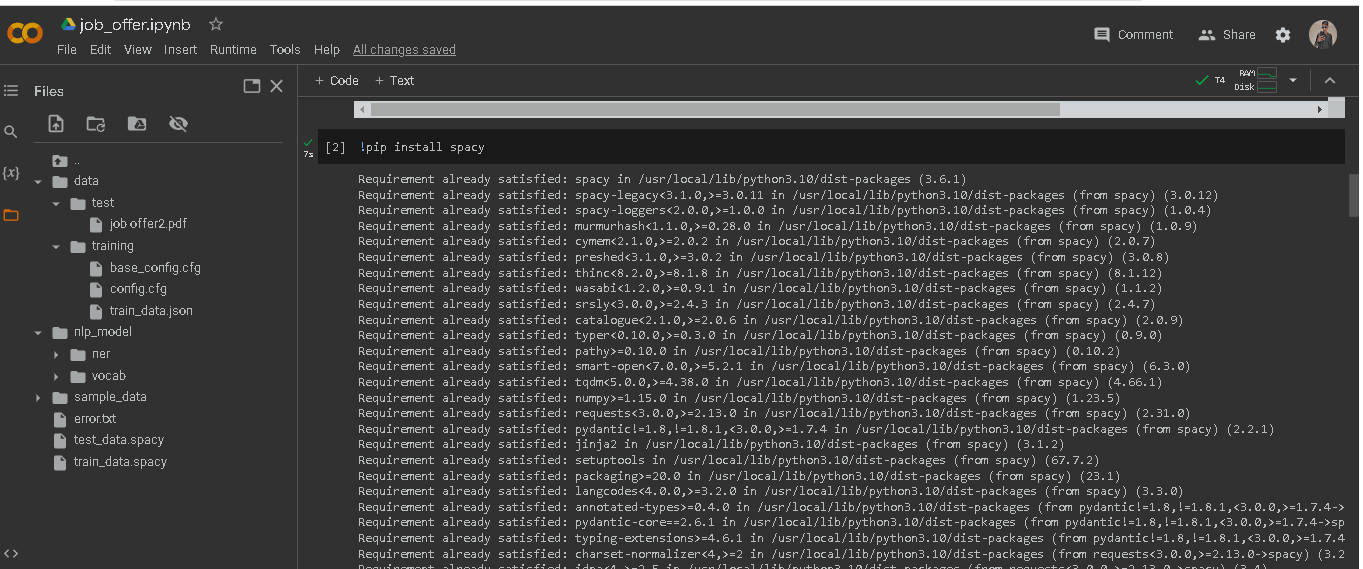
II. Steps of setting the project :

* 1. **Installations :**
* **Spacy\_transformation :**



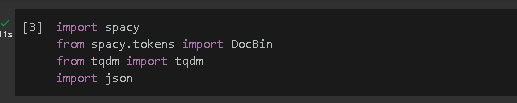
**“!pip install spacy\_transformers”** is used to install the "spacy\_transformers" package. This package is a valuable tool for incorporating transformer-based models, such as BERT, RoBERTa, and others, into the spaCy NLP framework. The primary purpose of this package is to enable spaCy to work seamlessly with transformer models

* **Spacy :**

****

**“!pip install spacy**” is used to install the "spacy" package. It serves as the foundation for building NER (Named Entity Recognition) models and performing various text analysis tasks in our project.

1. **Importing libraries :**

****

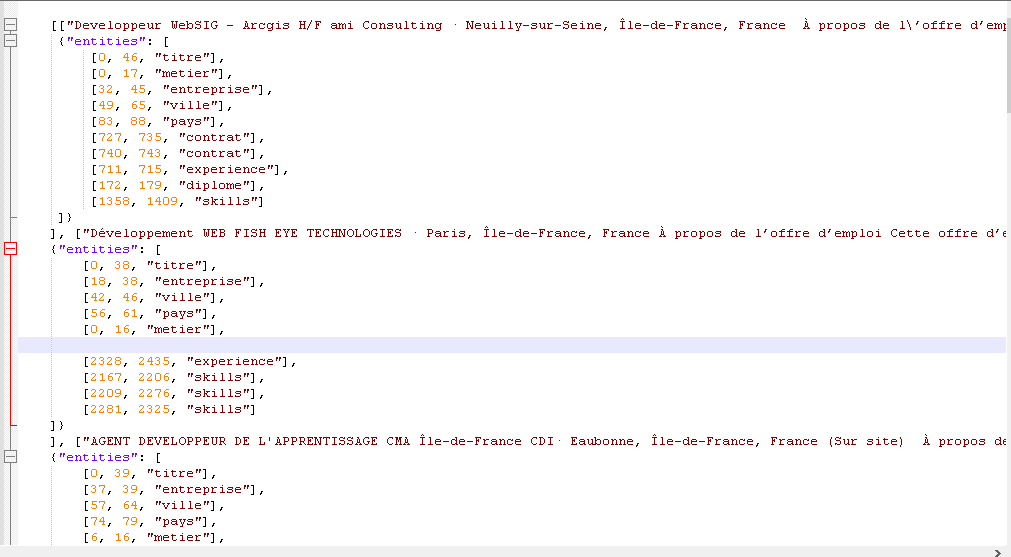
**spacy** is the core NLP library used for text processing and entity recognition, **DocBin** is used for efficient serialization of processed documents, **tqdm** enhances user experience by providing progress bars, and **json** is for working with structured data storage

1. **Choosing GPU as our runtime :**

Changing the runtime to GPU in Google Colab for our NER project is a choice to significantly accelerate our project's computations, reduce processing times, and enable the efficient training of complex models. It leverages the available hardware resources effectively and enhances our productivity in developing and experimenting with your NER system.

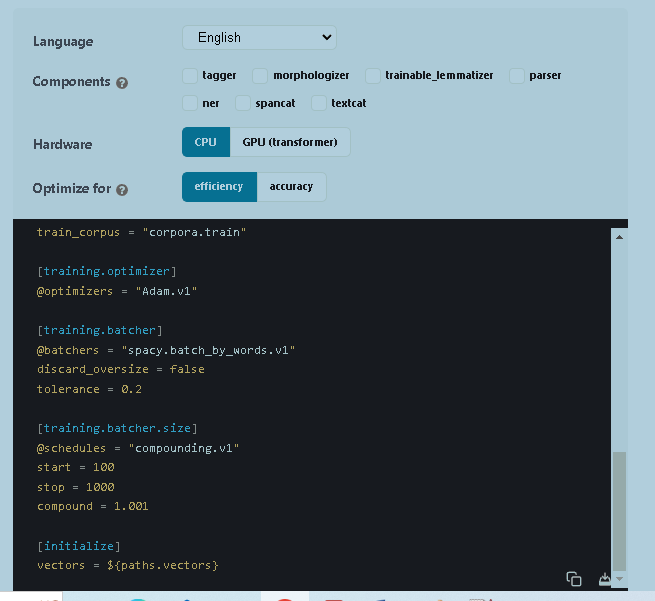
1. Preparing our project setup :

* **Loading our training data in the data folder :**

****

* **Loading base config file from the spacy library :**

The base\_config file from the spaCy library serves as a foundational blueprint for creating and customizing our NER model. It defines the model's architecture, hyperparameters, components, and other important settings.

****

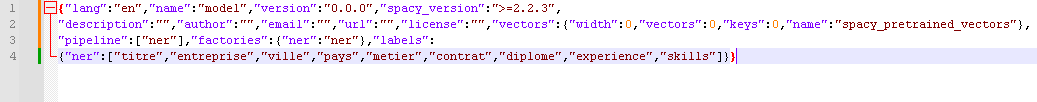
* **Loading our test data in the test folder :**

****

* **Loading our meta.json and tokenizer files in the nlp\_model folder :**

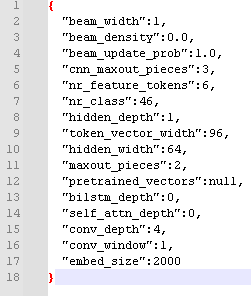
Meta.json :

the **meta.json** file contains metadata and configuration information for our spaCy NER model, including language, version, required spaCy version, pipeline components, and named entity labels.



* **Loading our cfg, model , moves files in the “ner” folder :**

Cfg:



The cfg file defines various hyperparameters and architectural settings for the neural network model used in our NER project. These settings influence how the model processes and learns from the input text data, ultimately impacting its ability to recognize named entities in text.

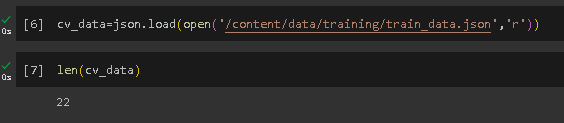
Model:

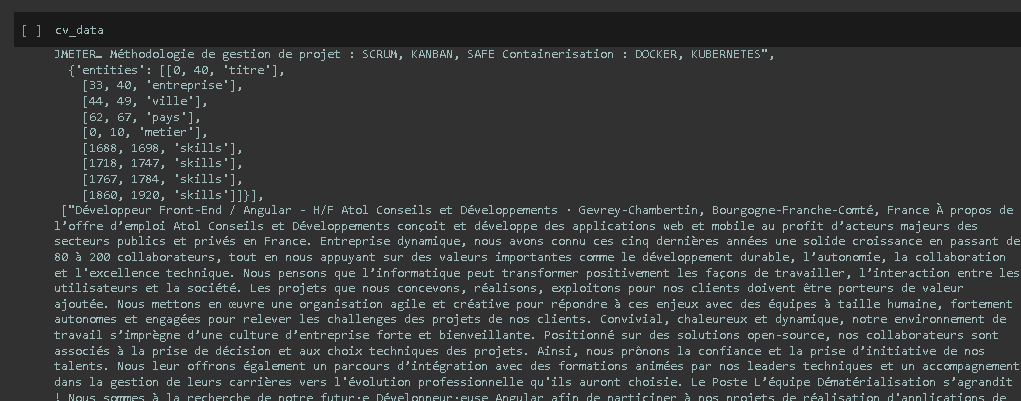
Utilize a pre-trained transformer model, in our case : RoBERTa, with a sequence tagging

* It provides rich contextual representations of text, enables efficient transfer learning, and ultimately contributes to the accuracy and robustness of your named entity recognition system for job advertisements
* **Loading strings.json , vectors , key2row, lexemes.bin , lookup.bin in our vocab folder:**

these files in our "vocab" folder are integral to our spaCy NLP model. They collectively manage the vocabulary, word vectors, and lexeme information necessary for efficient text processing, including Named Entity Recognition (NER). They enable spaCy to quickly retrieve linguistic information, perform semantic operations, and enhance the overall performance of our NER model.

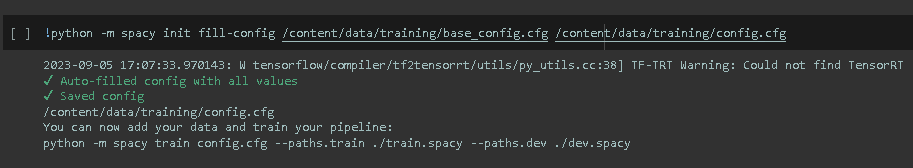
1. Reading our training data using json :





* **Loading our config.cfg file from spacy library :**

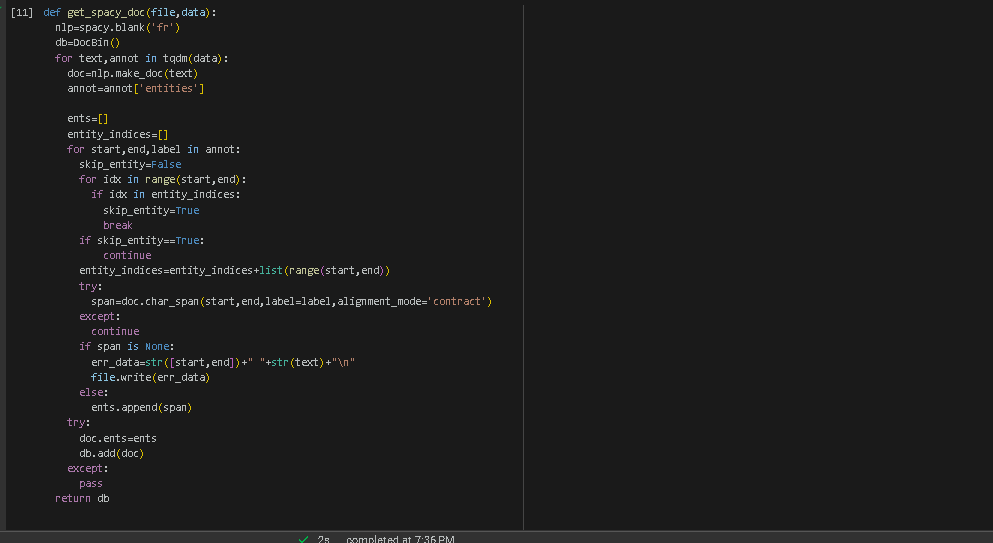
The command **!python -m spacy init fill-config** automates the process of creating a spaCy configuration file (**config.cfg**) based on a template (**base\_config.cfg**). It streamlines the initial setup of our NER project by providing a structured and customized configuration file that aligns with our project's data, hyperparameters, and model architecture

****

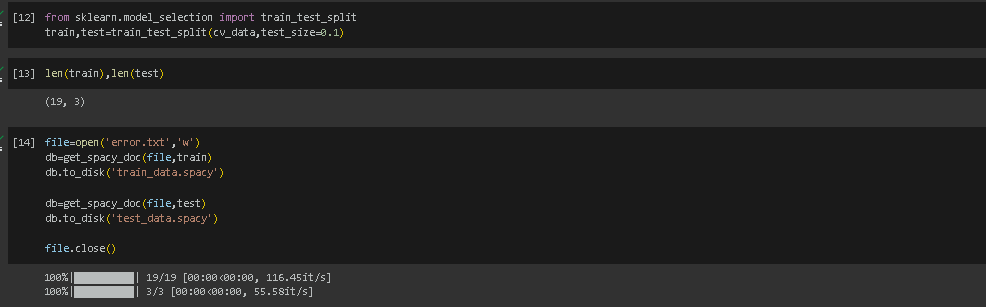
* With minor changes in the configurations , like changing the max\_epoch to 2000 instead of 20000 and max\_steps to 100

1. Using spacy custom training model in our project case :

This function is responsible for processing and converting our project's data into a spaCy **DocBin** object

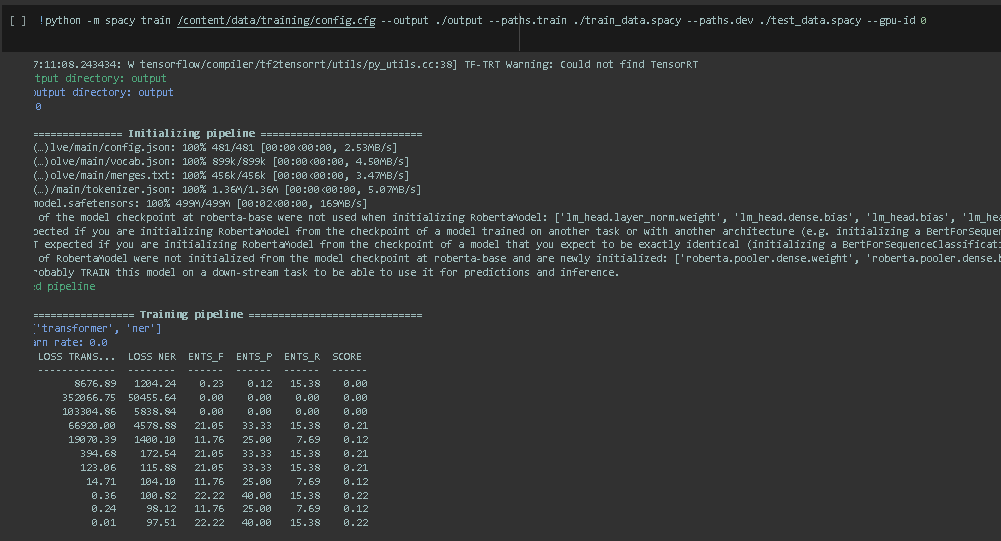


1. Splitting our data :

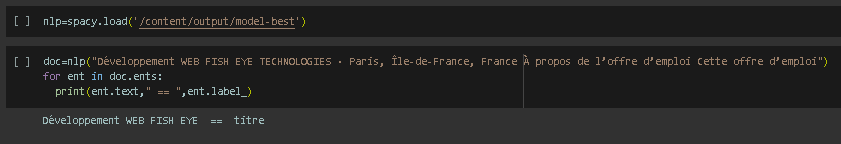


1. importing the train\_test\_split function from sklearn.model\_selection to divide our cv\_data into training and testing subsets.
2. Splitting the data into training and testing sets using train\_test\_split with a test size of 20%, and checking the lengths of the resulting subsets.
3. creating a new file named 'error.txt' in write mode to capture potential errors during data processing.
4. generating SpaCy DocBin objects, db, containing processed training and testing data using the get\_spacy\_doc function, and saving them to disk as 'train\_data.spacy' and 'test\_data.spacy'.
5. Finally, we close the 'error.txt' file to complete the data processing and error recording steps.
6. Training our data :

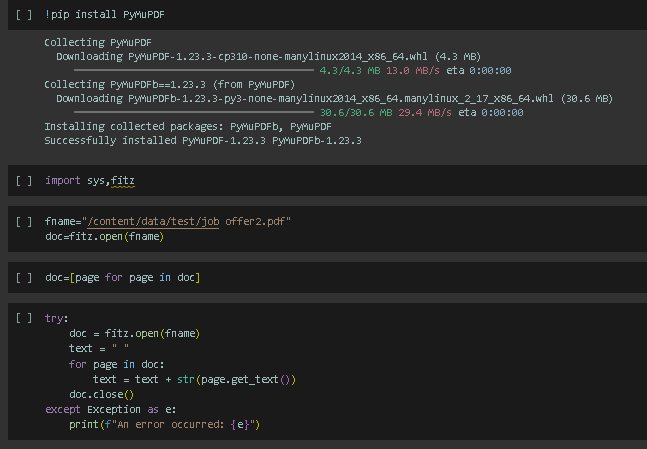
==> Employing SpaCy's command-line training interface to train a language model. It uses a configuration file (config.cfg) to define training settings. The trained model and results will be stored in the 'output' directory. Training and testing datasets, previously prepared as SpaCy DocBin files (train\_data.spacy and test\_data.spacy), are specified. If available, a GPU (ID 0) is utilized to expedite training. This command orchestrates the model training process, configuring data paths, GPU usage, and output directory.



1. Testing a sample :

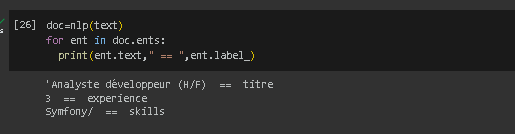


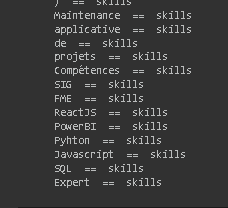
1. Testing the model using PDF library :



installing the PyMuPDF library for working with PDF documents and importing necessary modules, opens a PDF file specified by fname, extracts text content from each page, and aggregates it. If any error occurs during this process, an error message is printed. We will extract job offer text from PDF files, for testing the trained model's capabilities on real-world PDF data.

1. Output :





Even though the dataset is only 22 data we still managed to test the model and to get out with some valid data and we are missing some other data of “Ville” and “Pays” but with training the mode with much more data the model will improve and as a consequence the results will improve as well