L'intelligence artificielle comme outil pour accéder à l'emploi

Traitement de texte : prise en main des caractéristiques et premières mises en place de méthodes

Chems, Selmane et Indira
Simplon.co
02/06/2023

Introduction

L'objectif de ce rapport est de décrire les différentes méthodes mises en place pour travailler le sujet de traitement de texte pour répondre aux besoins de notre client.
Notre sujet est le « Historic Search Trend analysis ».

Dans une première partie, nous ferons une analyse de notre dataset afin de permettre de mieux expliciter nos choix en matière de modèle d'IA utilisé.

Dans une seconde partie, nous verrons comment nous avons intégrés cela sur une API/BDD.

Dans la troisième partie nous montrerons comment nous avons établi notre choix d'application Streamlit et ce que nous permettrons de faire grâce à elle sur le long terme. Notre but étant de créer un outil permettant d'aider les utilisateurs à trouver l'emploi de leurs rêves en ciblant les compétences requises généralement sur ce type de poste.

Nous analyserons en détail les problèmes pouvant intervenir avant de conclure sur nos recommandations quant à l'utilisation des différentes méthodes et caractéristiques pour la technique de traitement de texte.

1. Traitement de texte analyses préliminaires

Analyse base de données

Nous avons pris notre dataset sur Kaggle. Cet ensemble de données a extrait les offres d'emploi des résultats de recherche de Google pour les postes d'analyste de données aux États-Unis.

La collecte de données a commencé le 4 novembre 2022 et a ajouté quotidiennement environ 100 nouvelles offres d'emploi à cet ensemble de données. Nous avons démarré le projet le 09/05/2023. Notre datasets s'arrête donc à cette date.

Forme du dataset : (17977, 27)

Nous effectuons des manipulations et constatons que le dataset est extrêmement déséquilibré avec beaucoup de valeurs nulles. Nous décidons de commencer par le nettoyer.

Nettoyage des données

Comme pour la plupart des tâches d'analyse de texte, il est important de nettoyer les données pour éliminer les informations indésirables, les caractères spéciaux, les doublons, etc.

Voici les étapes suivies :

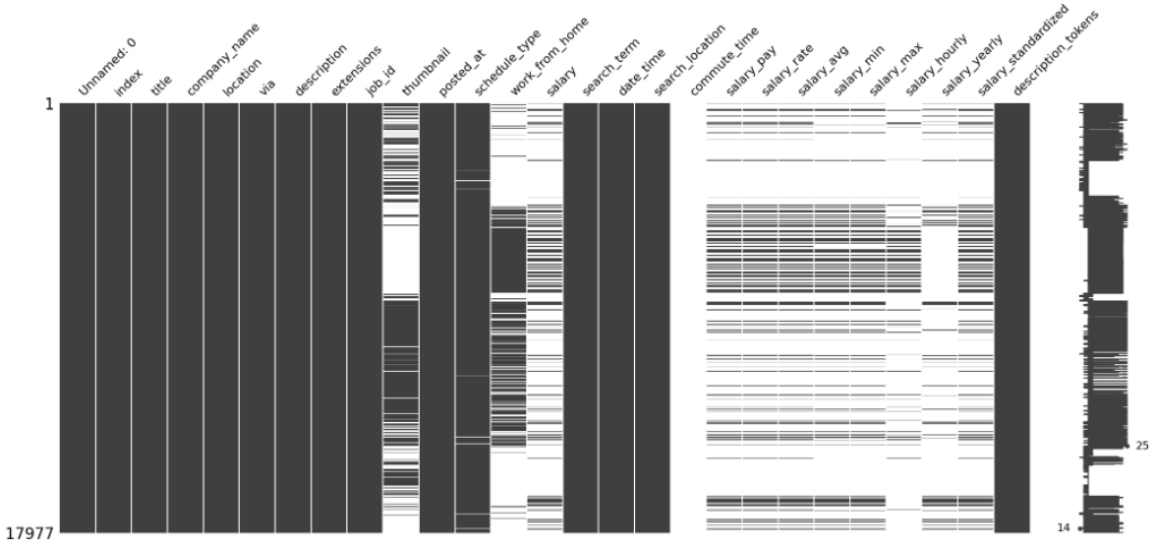
1. Suppressions des colonnes inutiles qui ne sont pas pertinentes pour votre analyse.

Nous effectuons une analyse afin de voir quelle est la proportion des colonnes nulles.

Figure 1 : Représentation en pourcentage des colonnes avec des valeurs nulles

	NaN Count	NaN Percentage
commute_time	17977	100.000000
salary_yearly	16525	91.923013
salary_hourly	15966	88.813484
salary_max	14720	81.882405
salary_min	14720	81.882405
salary	14508	80.703121
salary_standardized	14508	80.703121
salary_avg	14508	80.703121
salary_rate	14508	80.703121
salary_pay	14508	80.703121
work_from_home	9836	54.714357
thumbnail	9063	50.414418
schedule_type	115	0.639706
location	15	0.083440

Figure 2 : Représentation totale des colonnes avec les valeurs nulles en blanc



2. Vérification des valeurs manquantes (NaN) dans les colonnes et décidez comment les traiter (suppression des lignes, imputation, etc.).

Nous avons choisis de supprimer les valeurs nulles et d'exploiter uniquement les colonnes qui nous avons estimés être les plus utiles

3. Élimination des caractères spéciaux, les balises HTML ou tout autre contenu non pertinent qui peut affecter notre analyse.
4. Normaliser le texte en convertissant tout en minuscules, en supprimant la ponctuation, etc.

Traitement du texte

Traitement du texte : Pour obtenir des informations plus détaillées à partir du texte des descriptions d'emploi, nous avons utilisé différentes techniques de traitement du texte, telles que :

Tokenization : Division des descriptions d'emploi en mots individuels ou en n-grammes pour faciliter l'analyse.

Stop words removal : Suppression des mots courants (tels que "et", "le", "de", etc.) qui n'apportent pas de valeur informative.

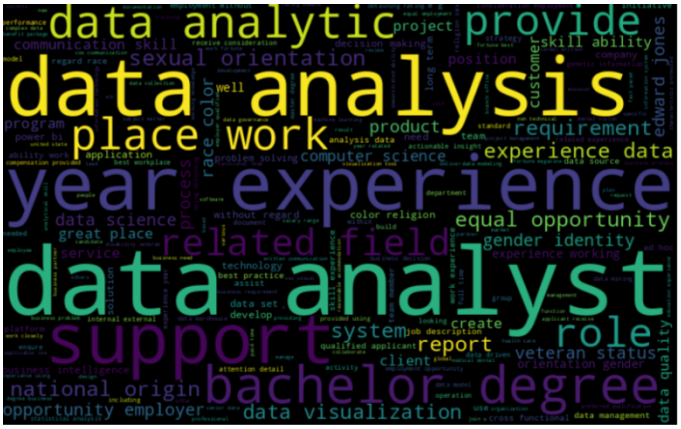
Lemmatization : Réduction des mots à leur forme de base pour regrouper les variantes (par exemple, "analyser", "analyse" deviennent "analyse").

Visualisation des données : Utilisation des graphiques et des visualisations pour rendre les résultats de votre analyse plus compréhensibles et interactifs.

Analyse exploratoire des données

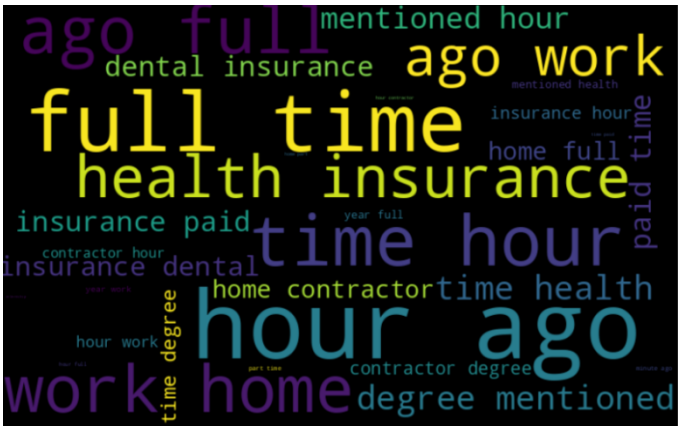
Une fois les données nettoyées, on fait une analyse exploratoire pour obtenir des informations générales sur les publications d'offres d'emploi.

Figure 3 : Visualisation de notre dataset colonne « description_tokens_lemmatized »



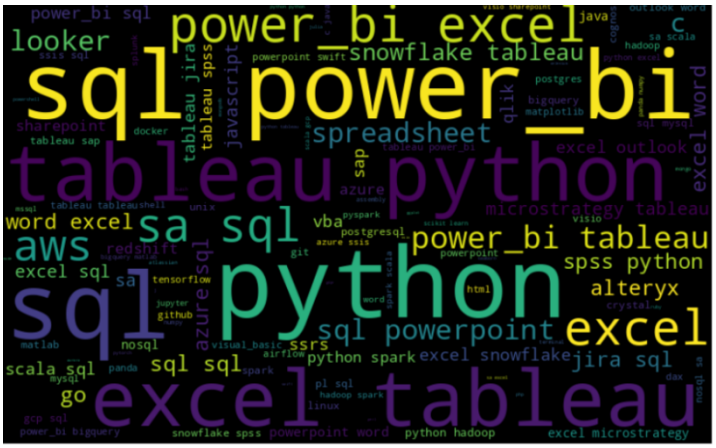
Nous observons ici les mots les plus fréquents qui expliquent les caractéristiques de l'emploi.

Figure 4 : Visualisation des mots les plus fréquents dans la colonne « extensions_tokens_lemmatized »



Nous observons ici les mots les plus fréquents qui expliquent les caractéristiques descriptives de l'emploi et ses avantages.

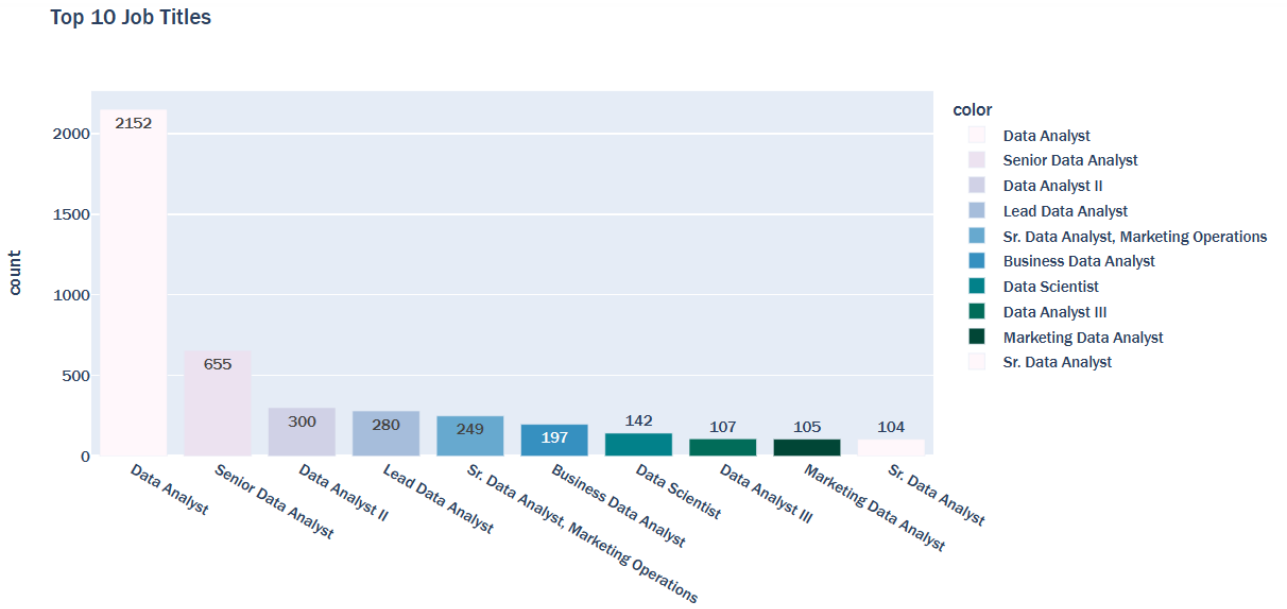
Figure 5 : Mots les plus fréquents dans la colonne description



Nous observons que les termes revenant le plus souvent sont :
Power_bi
Python
SQL
Tableau

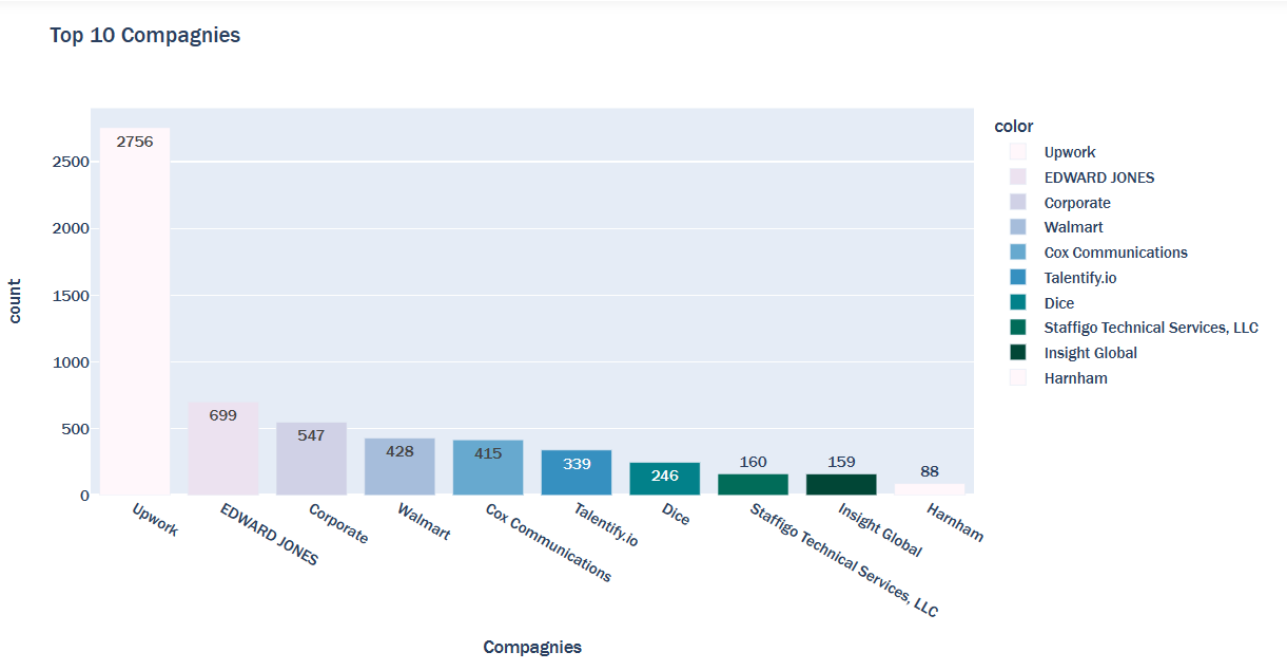
Suite à ce premier traitement nous regardons de plus prés les autres métriques du dataset.

Figure 1: Les 10 métiers les plus présents dans le dataset



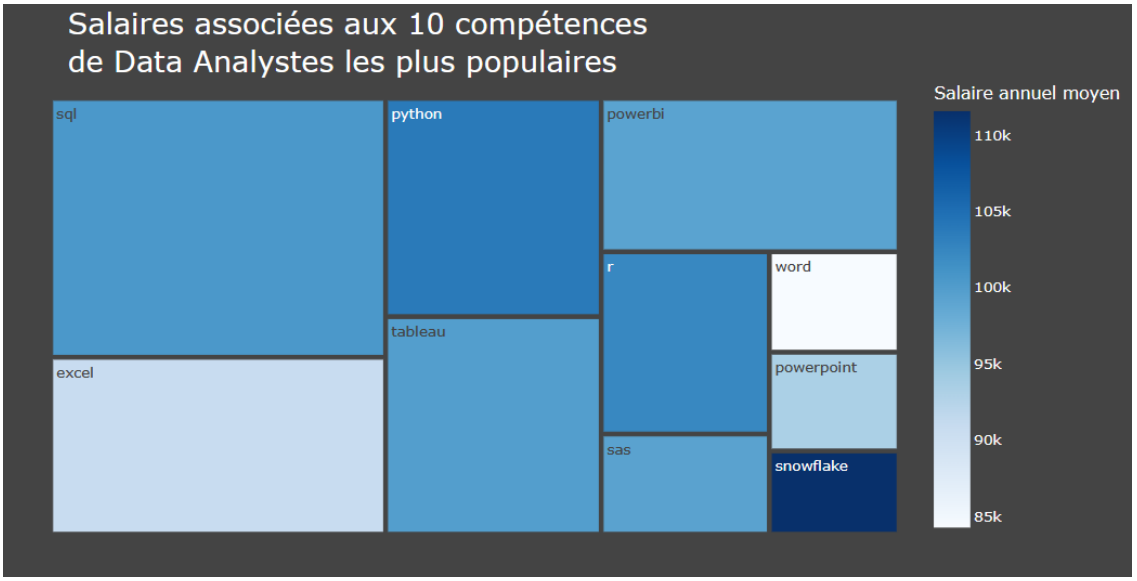
Nous remarquons que le métier de Data Scientiste est également présent malgré le fait que la recherche pour générer ce dataset est uniquement porté sur le métier de Data Analyste. Cela nous indique d'ores et déjà que certaines compétences sont communes aux deux métiers.

Figure 2: Le top 10 des compagnies par somme d'annonces



Le nom « Upwork » est une plateforme de missions orienté vers les freelances, ce qui explique leur sur-representation en matiére d'annonces.

Figure 3: Technologies les plus recherchées et leur salaires.



Même si nous avons fait le choix de supprimer les colonnes sur les salaires car il y avait près de 80% valeurs manquantes, ce graphique est important à étudier pour deux raisons. Premièrement cela nous indique suite à nos recherches que poster le salaire avec une offre de poste n'est pas une pratique courante aux USA.

Deuxièmement, cela nous permet d'observer que les compétences les plus demandées sont SQL et Excel. Mais les annonces qui demandent la maîtrise de SQL sont significativement mieux payées.

Python est une compétence très recherchée également et à des salaires élevés qui sont annoncés. Snowflake semble faire partie des technologies qui émergent et sont donc bien rémunérées.

2. Créer une API connecté à une BDD sur le Cloud Azure

Qu'est-ce qu'une API et pourquoi nous avons besoin?

Une API (Application Programming Interface) est un moyen de communication entre différentes applications logicielles. Elle permet à notre application de fournir des fonctionnalités ou d'accéder à des données d'autres applications, systèmes ou services.
Les API sont essentielles pour créer des applications interconnectées, car elles permettent le partage et l'échange de données de manière sécurisée et structurée.

Objectif de notre API :l'utilisateur doit être capable de rentrer un poste avec ses besoins spécifiques tel que : Data analyste en temps partiel et spécialisé en SQL.

Créer BDD vide sur Azure et la connecter avec MySQL Workbench

On accède au portail Azure et une fois dessus on clique sur "Créer une ressource" en haut à gauche de la page. On recherche "Base de données Azure MySQL flexible" dans la barre de recherche et on sélectionne l'option correspondante. Nous avons choisi « Azure Database for MySQL Flexible Server ».
Dans l'onglet "Basiques" du formulaire de création de ressource, nous remplissons les informations requises puis vérifions tout avant de créer la ressource.

Une fois la base de données créée, nous pouvons accéder à celle-ci en sélectionnant la ressource de base de données dans le portail Azure.

Nous allons ensuite trouver les informations de connexion, telles que l'adresse du serveur, le nom d'utilisateur et le mot de passe, dans la section "Chaîne de connexion" ou "Paramètres du serveur". C'est ce qui va nous servir pour la connexion avec MySQL Workbench.

Créer une API adapté à notre besoin

1 : Importer les modules nécessaires

Ici

```
import mysql.connector
from fastapi import FastAPI
import random
```

mysql.connector permet la connexion à MySQL et FastAPI pour créer l'API. Le module random sert à créer un ID random.

2 : Initialiser l'API ensuite avec `app = FastAPI()` Cela crée un FastAPI app objet

3 : On connecte avec les paramètres spécifiques pour la database MySQL

```
conn = mysql.connector.connect(
    user="    sdine",
    password="Ounissi     ",
    host="myserverchems.mysql.database.azure.com",
    port=3306,
    database="linkedin_bdd"
)
```

4 : Créer deux tables permettant de stocker et donner des prédictions

```python
table_1 = "Features"
table_2 = "Predictions"
```

5 : Fonction qui permet de récupérer les données de la BDD

```python
# Fonction pour récupérer les données depuis la base de données MySQL.
def get_data_from_database(table1, table2, cursor=cursor):
    cursor.execute(f"SELECT * FROM {table1}")
    data1 = cursor.fetchall()
    cursor.execute(f"SELECT * FROM {table2}")
    data2 = cursor.fetchall()
    return data1, data2
```

6 : Endpoint (l'extrémité d'un canal de communication) pour récupérer la data grâce à une requête GET

```python
@app.get("/data/get")
async def get_data():
    data = get_data_from_database(table_1="predictions")
    print("Message: Data retrieved successfully")
    return {"data": data}
```

Cet endpoint est actif quand la requête GET est lancée à data/get. Elle appelle get_data_from_database qui est une fonction de la table prédictions et qui ensuite retourne la data sous format JSON.
C'est mieux d'utiliser JSON car il est interprétable par tous les langages.

7 : On crée ensuite une fonction d'insertion dans le dataset qui permet de prendre un dictionnaire de data, établir la connexion et insérer la data spécifique dans la table

```python
# Fonction permettant d'insérer les données.
def insert_data_to_database(data, table1, table2, columns_features, connexion=cnx, cursor=cursor):
    try:
        value_features = list(data.values())
        del value_features[-1]
        table1_sql = f"INSERT INTO {table1} ({', '.join(columns_features)}) VALUES ({', '.join(['%s' for _ in range(len(columns_features))])})"
        cursor.execute(table1_sql, value_features)
        inserted_id = cursor.lastrowid
        table2_columns = ["id_fk", "y_pred"]
        table2_values = [inserted_id, data["Prediction"]]
        table2_sql = f"INSERT INTO {table2} ({', '.join(table2_columns)}) VALUES ({', '.join(['%s' for _ in range(len(table2_columns))])})"
        cursor.execute(table2_sql, table2_values)
        print("Données insérées avec succès.")
        connexion.commit()
    except Exception as e:
        print(e)
```

8 : Endpoint pour envoyer les données grâce à une requête POST. Elle appelle la fonction insert_data_to_database pour inserer la data reçu dans les tables « Features » et « Prédictions » et retourner un message si c'est un succès.

```python
# Route pour envoyer des données via une requête POST (before).
@app.post("/data/post")
async def send_data(data:dict):
    insert_data_to_database(
        data=data,
        table1="features",
        table2="predictions",
        columns_features=columns_features
        )
    return {"message": "Données insérées avec succès"}
```

Pour finir on va alimenter l'API via streamlit et pour ce faire nous allons voir comment développer une application streamlit approprié.

3. APP Streamlit

Nous avons donc créé notre API mais on doit se recentrer sur les raisons pour lesquelles elle a été créé et l'objectif que nous recherchons à remplir.

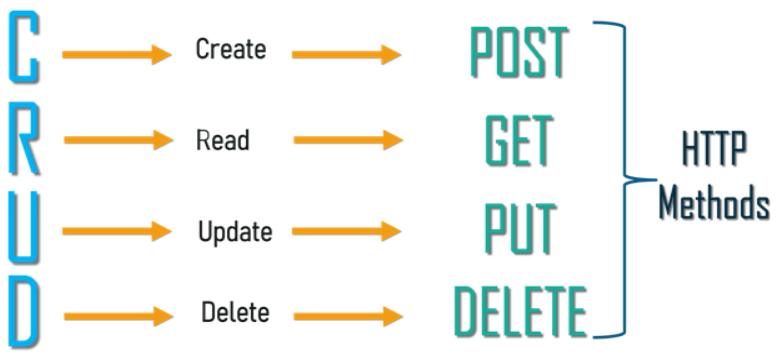
L'objectif de notre API: permettre aux utilisateurs d'entrer des mots-clés d'un type de poste qui les aide à aiguiller les compétences qu'ils devront acquérir.
Les fonctionnalités qu'on souhaite offrir aux utilisateurs : une application simple d'utilisation et spécifique aux métiers de data analyst.

a. Concevoir l'architecture de l'API

Dans le contexte de la création d'une API, le protocole couramment utilisé est REST (Representational State Transfer).
REST est un style d'architecture logicielle qui définit un ensemble de contraintes pour la création de services web.
L'acronyme CRUD est souvent utilisé en conjonction avec REST, car il représente les opérations de base sur les données : Create (Créer), Read (Lire), Update (Mettre à jour) et Delete (Supprimer).

Dans le cadre de notre API nous allons concevoir nos endpoints en suivant les principes REST et les opérations CRUD correspondantes :

Figure 1 : Méthodologie CRUD



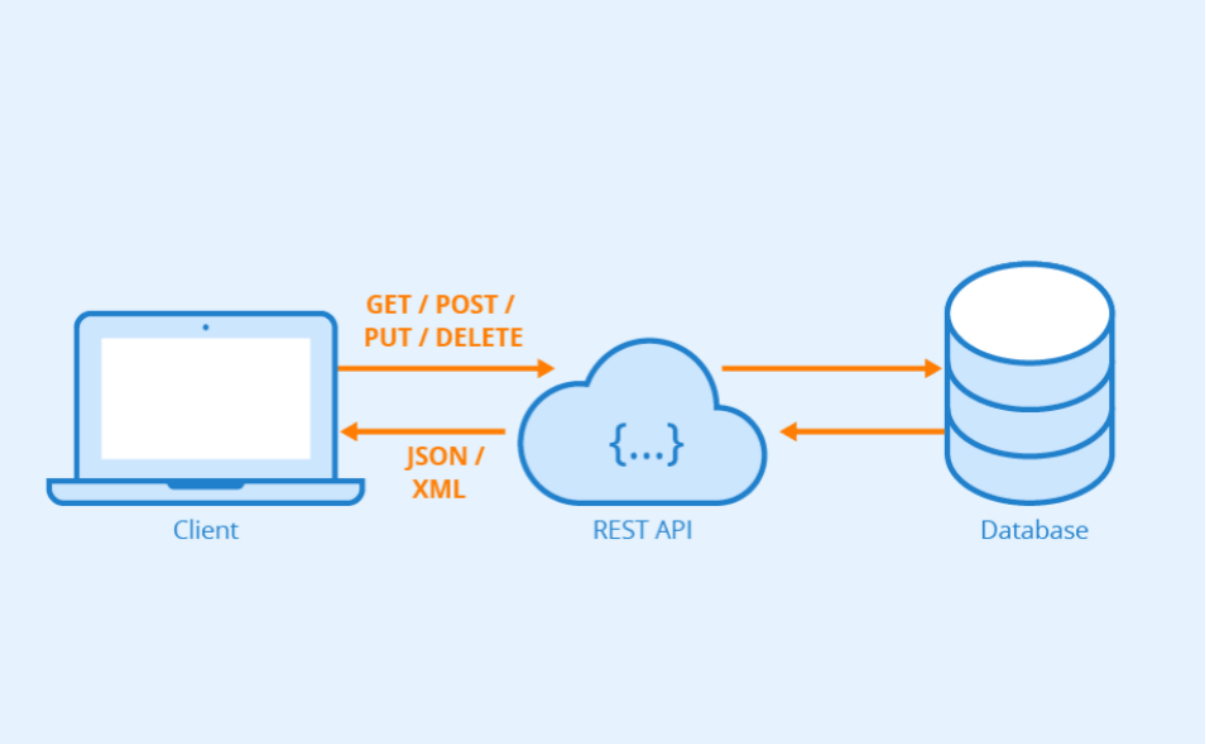
Créer (Create) : Utilisez la méthode HTTP **POST** pour envoyer des données au serveur et créer de nouvelles ressources. Cela signifie que nous allons avoir un endpoint comme **/data** pour créer de nouvelles données.

Lire (Read) : Utilisez la méthode HTTP **GET** pour récupérer des données existantes à partir du serveur. Cela sera un endpoint comme /**data/{id}** pour récupérer les données correspondant à un identifiant spécifique.

Mettre à jour (Update) : Utilisez la méthode HTTP **PUT** ou **PATCH** pour mettre à jour les données existantes sur le serveur. De la même manière, nous allons avoir un endpoint comme **/data/{id}** pour mettre à jour les données d'une ressource spécifique.

Supprimer (Delete) : Utilisez la méthode HTTP **DELETE** pour supprimer des ressources existantes du serveur. Ce sera un endpoint comme **/data/{id}** pour supprimer une ressource spécifique ou plusieurs d'entre elles.

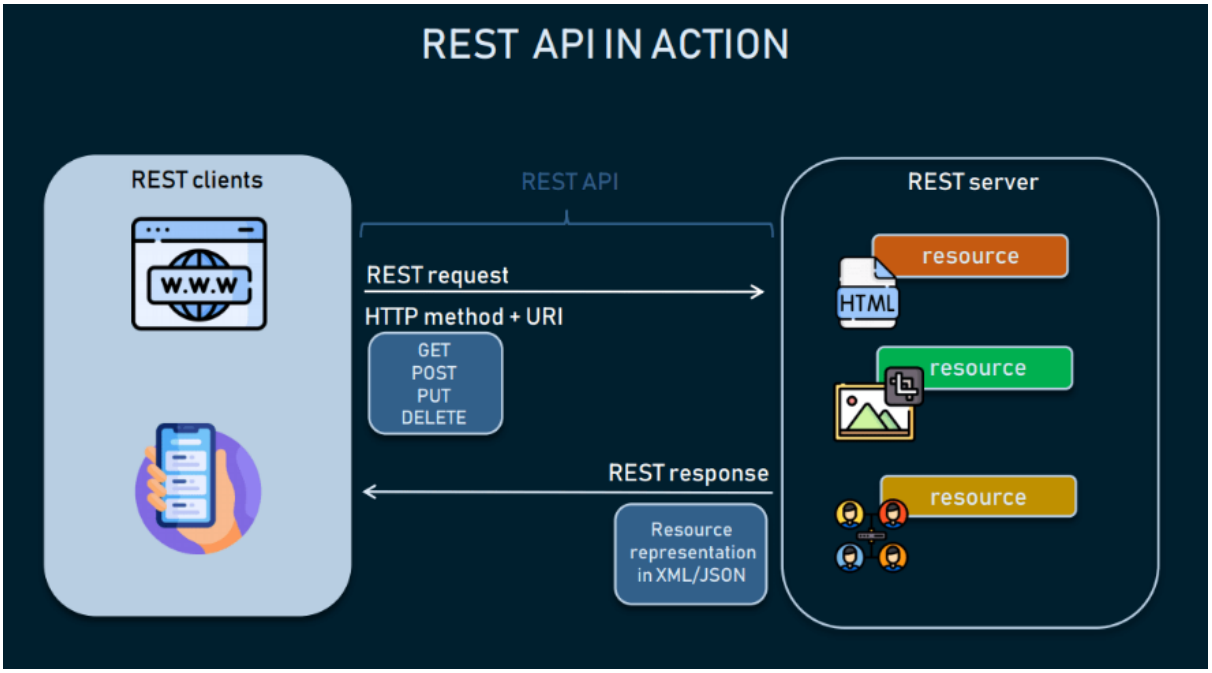
Figure 1 : Architecture d'une appli avec une API



b. Développer les endpoints de l'API

Implémentez les endpoints de votre API en fonction de l'architecture définie dans la partie deux.

Figure 2 : Visualisation d'une API en méthode REST



L'utilisateur se connecte à l'application et envoie une requête REST. Celle-ci se connecte à l'API qui envoie la demande vers la BDD. La data est ensuite renvoyé vers l'API qui l'envoie sous format JSON à l'application de l'utilisateur.

c. Tester l'API

Une fois que nous avons développé les endpoints, nous avons effectué des tests approfondis pour nous assurer que notre API fonctionne correctement. Nous avons testé différentes requêtes et nous nous sommes assuré que les réponses envoyées sont conformes aux attentes.

d. Documenter l'API

- Nous avons documenté clairement notre API pour aider les utilisateurs à comprendre comment l'utiliser.

e. Déployer l'API

- Choix plateforme d'hébergement appropriée pour déployer notre API. Il existe de nombreuses options, telles que des services cloud (AWS, Azure, Google Cloud), des serveurs dédiés ou des plateformes spécifiques pour les applications web (Heroku, Netlify, etc.).

4. Mise en place des méthodes d'ensemble pour la modélisation

Dans le domaine du traitement automatique des langues, l'allocation de Dirichlet latente (de l'anglais Latent Dirichlet Allocation) ou LDA est un modèle génératif probabiliste permettant d'expliquer des ensembles d'observations, par le moyen de groupes non observés, eux-mêmes définis par des similarités de données.

C'est le modèle que nous avons décidé de retenir après avoir testé un modèle avec T-SNE couplé avec le Word embedding.

a. Résultats avec la méthode

Extracting Topics using LDA in Python

```
# Récupérer la colonne "description" sous forme de liste
description_list = df_train["description"].tolist()
```

```
stemmer = SnowballStemmer("english")
```

```
'''
Write a function to perform the pre processing steps on the entire dataset
'''
def lemmatize_stemming(text):
    return stemmer.stem(WordNetLemmatizer().lemmatize(text, pos='v'))

# Tokenize and lemmatize
def preprocess(text):
    result=[]
    for token in gensim.utils.simple_preprocess(text) :
        if token not in gensim.parsing.preprocessing.STOPWORDS and len(token) > 3:
            result.append(lemmatize_stemming(token))

    return result
```

```
# Créer une liste pour stocker les résultats
processed_text = []

# Appliquer la fonction lemmatize_stemming à chaque élément de la liste description_list
for text in description_list:
    processed_text.append(preprocess(text))
```

Voici une explication synthétique de chaque partie du code :

Prétraitement des données : Le code commence par récupérer la colonne "description" du jeu de données d'entraînement et la stocke sous forme de liste dans la variable description_list. Ensuite, un stemmer (SnowballStemmer) est créé pour effectuer la lemmatisation des mots en anglais.

Fonctions de prétraitement : La fonction lemmatize_stemming(text) est définie pour effectuer la lemmatisation et le stem des mots en utilisant le stemmer créé précédemment.

La fonction preprocess(text) est définie pour effectuer les étapes de prétraitement sur le texte. Les mots sont tokenisés et convertis en minuscules.

Les mots qui font partie des stopwords (mots courants qui n'apportent pas de sens) sont filtrés. Les mots de moins de 3 caractères sont filtrés. Les mots restants sont lemmatisés et stemmés en utilisant la fonction lemmatize_stemming.

Une liste vide, processed_text, est créée pour stocker les résultats du prétraitement.

La fonction preprocess est appliquée à chaque élément de la liste description_list et les résultats sont ajoutés à processed_text.

Création du dictionnaire : Un dictionnaire est créé à partir des données prétraitées (processed_text) en utilisant la classe corpora.Dictionary de la bibliothèque Gensim.

Le dictionnaire est affiché pour vérification.

Filtrage des mots rares et courants : Les mots qui apparaissent moins de 15 fois ou dans plus de 10% de tous les documents sont filtrés du dictionnaire en utilisant la méthode filter_extremes du dictionnaire.

Modélisation LDA avec Bag-of-Words

```
'''
OPTIONAL STEP
Remove very rare and ver common words:

- words appearing less than 15 times
- words appearing in more than 10% of all documents
'''
dictionary.filter_extremes(no_below=15, no_above=0.1, keep_n= 100000)
```

```
'''
Create the Bag-of-words model for each document i.e for each document we create a dictionary reporting how many
words and how many times those words appear. Save this to 'bow_corpus'
'''
bow_corpus = [dictionary.doc2bow(doc) for doc in processed_text]
```

```
'''
Preview BOW for our sample preprocessed document
'''
document_num = 20
bow_doc_x = bow_corpus[document_num]

for i in range(len(bow_doc_x)):
    print("Word {} (\"{}\") appears {} time.".format(bow_doc_x[i][0],
                                                     dictionary[bow_doc_x[i][0]],
                                                     bow_doc_x[i][1]))
```

Le modèle LDA est entraîné sur le corpus Bag-of-Words en utilisant la classe models.LdaMulticore de Gensim.

Les paramètres spécifiés pour l'entraînement du modèle comprennent le nombre de sujets (num_topics), le dictionnaire (id2word), le nombre de passes (passes), et le nombre de travailleurs (workers).

Running LDA using Bag of Words

```
# LDA mono-core -- fallback code in case LdaMulticore throws an error on your machine
# lda_model = gensim.models.LdaModel(bow_corpus,
#                                    num_topics = 10,
#                                    id2word = dictionary,
#                                    passes = 50)

# LDA multicore
'''
Train your lda model using gensim.models.LdaMulticore and save it to 'lda_model'
'''
# TODO
lda_model =  gensim.models.LdaMulticore(bow_corpus,
                                        num_topics = 8,
                                        id2word = dictionary,
                                        passes = 10,
                                        workers = 2)
```

```
'''
For each topic, we will explore the words occuring in that topic and its relative weight
'''
for idx, topic in lda_model.print_topics(-1):
    print("Topic: {} \nWords: {}".format(idx, topic ))
    print("\n")
```

Affichage des sujets : Les mots et les poids associés à chaque sujet sont affichés à l'aide de la méthode print_topics du modèle LDA.

Classification des sujets : Une section facultative fournit des suggestions de noms pour chaque sujet basées sur les mots et les poids correspondants.

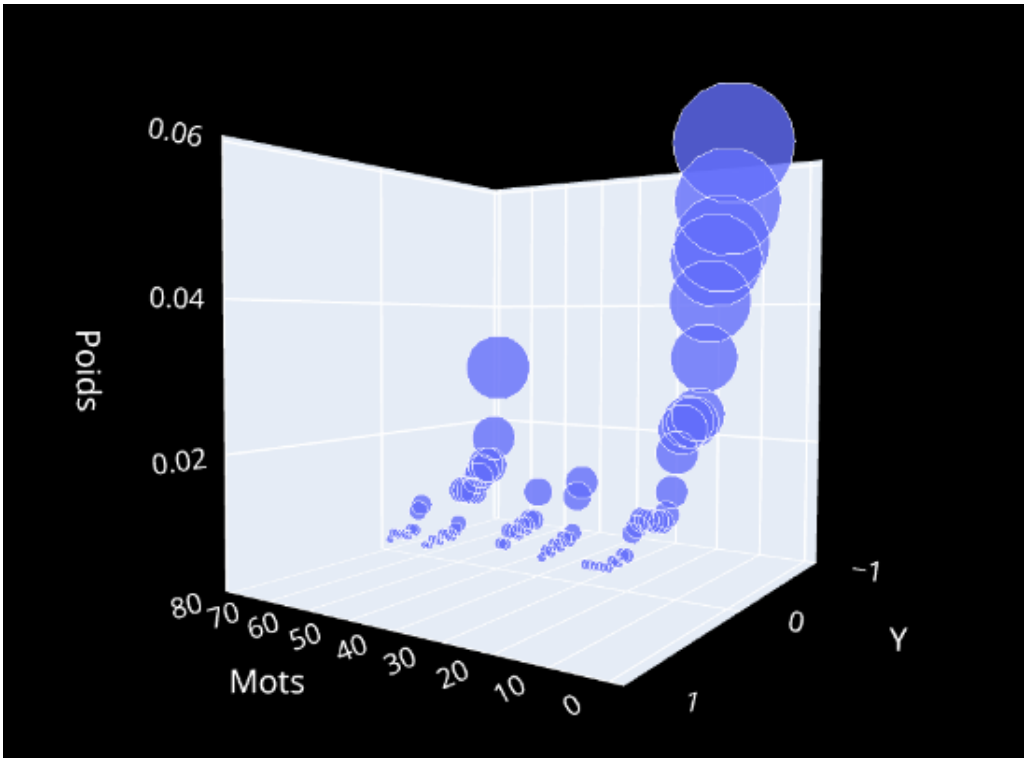
Visualisation : Le code utilise la bibliothèque Plotly Express pour créer un DataFrame contenant les informations sur les sujets.

Vizualisation

```
import plotly.express as px
```

```
# Créer un dataframe avec les informations des topics
topics_df = pd.DataFrame({
    'Topic': ["Carrières dans la publication et l'édition", 'Assurance et remboursement médical',
              'Marketing et publicité en ligne', 'Éducation et études universitaires',
              'Recherche médicale et scientifique', 'Logistique et transport',
              'Informatique et résolution de problèmes', 'Finance et gestion des actifs'],
    'Words': [
        'jone, edward, fortun, firm, branch, rank, magazin, publish, rat, obtain',
        'claim, centen, clinic, purchas, reimburs, elev, stock, pharmacie, root, tuition',
        'brand, media, websit, campaign, event, looker, consum, love, tech, tulsa',
        'univers, student, school, studi, survey, colleg, check, academ, institut, date',
        'clinic, suppli, machin, chain, patient, walmart, predict, scientist, algorithm, energi',
        'resum, agenc, capac, day, transport, send, forward, ongo, orchestr, readi',
        'oracl, map, file, troubleshoot, server, agil, resolv, flow, vendor, travel',
        'vaccin, bank, covid, audit, citi, asset, investig, architectur, azur, legal'
    ],
    'Weight': [
        [0.059, 0.052, 0.047, 0.045, 0.040, 0.033, 0.026, 0.025, 0.025, 0.024],
        [0.021, 0.016, 0.013, 0.012, 0.012, 0.012, 0.012, 0.012, 0.011, 0.010],
        [0.007, 0.007, 0.006, 0.006, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005],
        [0.016, 0.014, 0.009, 0.008, 0.008, 0.007, 0.007, 0.006, 0.006, 0.005],
        [0.014, 0.010, 0.010, 0.009, 0.009, 0.008, 0.008, 0.008, 0.006, 0.006],
        [0.031, 0.021, 0.017, 0.017, 0.016, 0.015, 0.013, 0.013, 0.013, 0.013],
        [0.008, 0.007, 0.006, 0.006, 0.006, 0.005, 0.005, 0.005, 0.004, 0.004],
        [0.010, 0.009, 0.006, 0.006, 0.005, 0.005, 0.005, 0.005, 0.005, 0.004]
    ]
})
```

Les informations sur les sujets sont fournies sous forme de listes pour les noms des sujets, les mots et les poids correspondants.



Le résultat fourni n'est pas très pertinent et ne sera donc pas retenu.

b. Analyse des résultats

Le DataFrame topics_df est créé avec les informations des sujets.
Les résultats peuvent être visualisés pour examiner les sujets et leurs mots importants.

Visualisation des topics et des poids : Le code suivant utilise la les informations extraites du dataset pour créer une visualisation des topics et de leurs poids.

Figure 4: Visualisation topics

```
Topic: 0
Words: 0.020*"clinic" + 0.017*"vaccin" + 0.013*"covid" + 0.012*"patient" + 0.006*"safeti" + 0.006*"clearanc" + 0.006*"investig"
+ 0.006*"militari" + 0.006*"studi" + 0.005*"claim"


Topic: 1
Words: 0.012*"oracl" + 0.010*"server" + 0.009*"azur" + 0.009*"map" + 0.007*"elev" + 0.007*"architectur" + 0.006*"dice" + 0.005
*"logic" + 0.004*"store" + 0.004*"titl"


Topic: 2
Words: 0.037*"centen" + 0.029*"claim" + 0.026*"reimburs" + 0.023*"purchas" + 0.022*"tuition" + 0.021*"root" + 0.020*"coverag" +
0.019*"cod" + 0.019*"stock" + 0.018*"dress"


Topic: 3
Words: 0.009*"walmart" + 0.007*"machin" + 0.007*"exampl" + 0.007*"retail" + 0.006*"scientist" + 0.006*"predict" + 0.006*"outli
n" + 0.005*"pipelin" + 0.004*"case" + 0.004*"looker"


Topic: 4
Words: 0.020*"resum" + 0.015*"campaign" + 0.014*"day" + 0.014*"agenc" + 0.012*"well" + 0.012*"capac" + 0.011*"send" + 0.011*"fo
rward" + 0.010*"ongo" + 0.010*"invest"


Topic: 5
Words: 0.014*"bank" + 0.009*"audit" + 0.008*"energi" + 0.007*"agil" + 0.006*"convers" + 0.006*"citi" + 0.006*"price" + 0.006*"c
redit" + 0.006*"regulatori" + 0.005*"resolv"


Topic: 6
Words: 0.007*"univers" + 0.007*"student" + 0.005*"file" + 0.005*"check" + 0.004*"suppli" + 0.004*"onlin" + 0.004*"school" + 0.0
04*"survey" + 0.004*"travel" + 0.004*"demand"
```

Figure 5: Les topics principaux

- Topic 0: Carrières dans la publication et l'édition
- Topic 1: Assurance et remboursement médical
- Topic 2: Marketing et publicité en ligne
- Topic 3: Éducation et études universitaires
- Topic 4: Recherche médicale et scientifique
- Topic 5: Logistique et transport
- Topic 6: Informatique et résolution de problèmes

c. Améliorations mises en place

Nous avons décidé de faire du séquençage de données suite à notre LDA.
Voici la manière dont on s'y est pris :

Figure 7: Partie séquence

```
# Preprocessing
text_data = df_v3['description'].values
input_sequences = []
target_sequences = []

# Prepare input and target sequences
for (schedule_type, search_term, search_location, description_tokens, year, month, job_offer) in zip(schedule_type_data, s
    input_sequence = f"{schedule_type} {search_term} {search_location} {description_tokens} {str(year)} {str(month)}"
    target_sequence = f"<start> {job_offer} <end>"
    input_sequences.append(input_sequence)
    target_sequences.append(target_sequence)

# Initialize Tokenizer
tokenizer = Tokenizer()

# Fit Tokenizer on data
tokenizer.fit_on_texts(input_sequences + target_sequences)

# Convert sequences to tokenized sequences
input_sequences = tokenizer.texts_to_sequences(input_sequences)
target_sequences = tokenizer.texts_to_sequences(target_sequences)

# Find maximum lengths
max_sequence_length = max([len(seq) for seq in input_sequences + target_sequences])

# Pad sequences
input_sequences = pad_sequences(input_sequences, maxlen=max_sequence_length, padding='post')
target_sequences = pad_sequences(target_sequences, maxlen=max_sequence_length, padding='post')

# Get the vocabulary size
vocab_size = len(tokenizer.word_index) + 1

# Define the input layers
encoder_inputs = Input(shape=(max_sequence_length,))
decoder_inputs = Input(shape=(max_sequence_length-1,))

# Define model architecture
latent_dim = 6 # Dimensionality of the latent space
embedding = Embedding(vocab_size, latent_dim)

encoder_embedding = embedding(encoder_inputs)
encoder_lstm = LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder_lstm(encoder_embedding)
encoder_states = [state_h, state_c]

decoder_embedding = embedding(decoder_inputs)
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_embedding, initial_state=encoder_states)
decoder_dense = Dense(vocab_size, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)

# Create the model
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)

# Compile the model
model.compile(optimizer='rmsprop', loss='sparse_categorical_crossentropy')

# Train the model
target_sequences_input = target_sequences[:, :-1]
target_sequences_output = target_sequences[:, 1:]
model.fit([input_sequences, target_sequences_input], target_sequences_output, epochs=5, batch_size=8)
```

On entraine un modèle de Deep Learning séquence to séquence pour générer un texte.

Figure 8: Les epochs liés au modèle

```
625/625 [==============================] - 5803s 9s/step - loss: 8.1747
Epoch 2/5
625/625 [==============================] - 5688s 9s/step - loss: 3.8347
Epoch 3/5
625/625 [==============================] - 5733s 9s/step - loss: 1.3985
Epoch 4/5
625/625 [==============================] - 5685s 9s/step - loss: 1.0253
Epoch 5/5
625/625 [==============================] - 5775s 9s/step - loss: 0.9620
```

On voit que le temps d'entrainement est très long.

Figure 9: La génération de texte

```python
# Generate text
encoder_model = Model(encoder_inputs, encoder_states)

decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]

decoder_outputs, state_h, state_c = decoder_lstm(decoder_embedding, initial_state=decoder_states_inputs)
decoder_states = [state_h, state_c]
decoder_outputs = decoder_dense(decoder_outputs)
decoder_model = Model([decoder_inputs] + decoder_states_inputs, [decoder_outputs] + decoder_states)

def generate_text(input_sequence):
    states_value = encoder_model.predict(input_sequence)
    target_sequence = np.zeros((1, 1))  # Start with empty target sequence
    confidence_threshold = 0.00011

    stop_condition = False
    generated_text = []

    while not stop_condition:
        output_tokens, h, c = decoder_model.predict([target_sequence] + states_value)
        sampled_token_index = np.argmax(output_tokens[0, -1, :])

        # Check if the sampled token index exists in the vocabulary
        if sampled_token_index in tokenizer.index_word:
            sampled_word = tokenizer.index_word[sampled_token_index]
            generated_text.append(sampled_word)
            print(np.max(output_tokens))
        else:
            # Handle the case when the token index is not found
            remaining_indices = set(range(len(tokenizer.index_word))) - {0}  # Exclude the unknown token
            sampled_token_index = np.random.choice(list(remaining_indices))
            sampled_word = tokenizer.index_word[sampled_token_index]
            generated_text.append(sampled_word)
            print(np.max(output_tokens))

        # Update the stop condition based on the generated token
#       if sampled_word == '<end>' or np.max(output_tokens) < confidence_threshold:
        if sampled_word == '<end>' or len(generated_text) > 100:
            stop_condition = True

        target_sequence = np.array([[sampled_token_index]])  # Update the target sequence
        states_value = [h, c]

    return ' '.join(generated_text)
```

Figure 10: Le texte issu du modèle

```
3.2730282E-05
httpsdhrcoloradogovdhrresourcesstudentloanforgivenessprograms 3¾ 58781 supervisor travelobsessed 5153034654 seicmm i470 projec
tsorganization bcforward 230 push directly remotevirtual auditboard shoe dish death 6079500 be later uploadfor dissimilar dai
attributesspecs smoke arcgis buildupdate utiliserez chda download 172500 onpage analystdashboard falcon etd galileo 68105 denk
en pave multisystem centrone knowledgeforce ricards texas quoting cultivates conceptdraw deliverablesassets remi développerez
testimonial fyii globaldata typefulltime serverside lijv1 2622400 fy21 azentas serf postdeployment fairer documentary effectiv
e singledose httpssyscobenefitscom ups exceed prioritized 3800 fullstory yuma 147700 wwwalphasensecom selector potato benefit
foundational howell leisure utah gb fails storyline presenting timecontract rei recognizable changesupdates 134 systemssoftwar
ehardware payback bringer encompasses subscriptionbased warner xmljsoncsv plusexcellent prewritten antispam
```

Conclusion

Cependant, la NLP doit faire face à plusieurs défis majeurs. Parmi ces défis figurent l'ambiguïté inhérente au langage humain, la complexité de la grammaire et de la syntaxe, ainsi que la nécessité de disposer de grandes quantités de données pour former des modèles efficaces. Un autre défi majeur réside dans le biais potentiellement présent dans les ensembles de données linguistiques, ce qui peut entraîner des modèles biaisés.

La prise en compte de ces défis revêt une importance cruciale pour le développement continu et le succès de la NLP. Dans le cadre spécifique de notre projet, nous avons rencontré le défi du changement continu dans le domaine du data analyst. En effet, cette filière est en constante évolution, ce qui se traduit par l'émergence de nouveaux titres de poste tels que "Data Engineer" ou "Data Steward", visant à établir une division des tâches plus précise en termes de compétences requises.

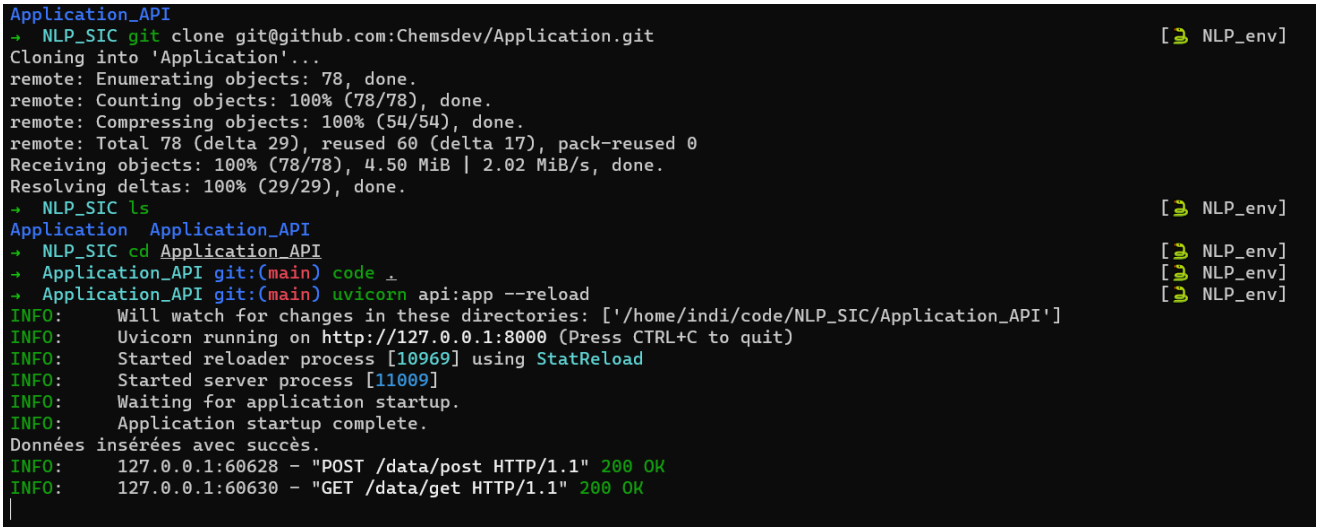
Cependant, il s'agit d'un processus continu, et pour obtenir les meilleures prédictions possibles à ce niveau, il est essentiel de suivre ce processus à long terme afin de segmenter les métiers de manière adéquate correspondant à leurs compétences spécifiques et leur stack technologique. Il convient de noter que la division du travail n'est souvent pas seulement le produit d'un besoin, mais également celui d'un changement sociétal. La présence de l'informatique et de l'intelligence artificielle est susceptible de générer de nombreux changements à venir.

Il convient de souligner que la division du travail n'est pas seulement le résultat d'un besoin, mais également d'un changement sociétal. Avec la présence croissante de l'informatique et de l'intelligence artificielle, il est probable que nous continuerons à connaître de nombreux changements dans ce domaine.

Ainsi, en combinant les efforts visant à résoudre les défis de la NLP, à suivre de près l'évolution des compétences requises dans le domaine de la data analyse, et à anticiper les changements futurs induits par l'informatique et l'intelligence artificielle, nous pourrons mieux préparer les professionnels de la data analyse à relever les défis de l'emploi et à saisir les opportunités qui se présentent.

Annexe guide utilisateur pour lancer l'application à partir du terminal.

Lancer API à partir du Terminal



Lancer l'APP Streamlit apres l'API (nouvel onglet terminal)