Projet Python
My first chatBot

Consignes & Informations générales :

⇒ Ce projet est à réaliser exclusivement en langage Python

⇒ Pièces jointes au sujet du projet :
- Fichiers des discours de présidents

⇒ Organisation des équipes :
- Ce projet est à réaliser en binôme (**un seul trinôme est autorisé** si un **nombre impair** d'élèves)
- La liste des équipes est à remettre aux enseignants **au plus tard** à la fin de la première séance de suivi de projet

⇒ Dates clé :
- Date de publication : **04/11/2023**
- Date de suivi 1 : Semaine du **13/11/2023**
- Date du premier dépôt : **26/11/2023** à **23h59**
- Date de suivi 2 : Semaine du **27/11/2023**
- Date du dépôt final : **17/12/2023** à **23h59**
- Date de soutenance : Semaine du **18/12/2023**

⇒ Rendu final :
- Le code du projet contenant les fichiers **.py** et **.txt**
- Le rapport en **.pdf**
- Un fichier **README.txt** donnant la liste des programmes et comment les utiliser en pratique. Ce fichier doit contenir toutes les instructions nécessaires à l'exécution ; il est très important pour expliquer à l'utilisateur comment se servir de l'outil.

⇒ Dépôt du projet :
- Le dépôt du projet se fera sous Moodle

⇒ Évaluation
- Barème détaillé : sera fourni plus tard
- Note finale du projet = Note code + Note rapport + Note soutenance
- Rappel : note projet = 20% de la note du cours « Programmation Python »
- Les membres d'une même équipe peuvent avoir des notes différentes en fonction des efforts fournis dans la réalisation de ce projet.

⇒ **Travail collaboratif**
- Le suivi du travail au sein d'une équipe projet se fera via Git
- Le tutoriel d'utilisation de Git sera mis en ligne la semaine du **06/11/2023**

Organisation du code :

⇒ La notation du code tiendra compte principalement de :
- L'implémentation des fonctionnalités demandées : avancer au mieux mais PAS en hors sujet
- La qualité du code fourni : organisation en fonctions, **commentaires**, noms de variables significatifs, respect des noms de fichiers.
- Ergonomie : Facilité d'utilisation de l'interface utilisateur

Notions pédagogiques traitées :

⇒ La réalisation de ce projet vous aidera à appliquer les notions pédagogiques suivantes :
- Notions des bases, listes 1D, listes 2D, les fonctions, les fichiers et les collections

⇒ Pour vous aider à la réalisation de ce projet, vous pouvez vous appuyer sur :
- Les supports du cours TI101 (I, B, P)
- Livres, cours divers sur le web. **ALERTE PLAGIAT !!** il ne s'agit pas de copier des programmes entiers.
- Enseignants **Efrei** lors des séances de suivi du projet

Description

Le projet que nous vous proposons porte sur l'analyse de texte. Ce projet vous permettra de comprendre quelques concepts de base utilisés dans le traitement de texte et de vous aider à comprendre l'une des méthodes utilisées pour développer des chatbot et/ou des intelligences artificielles génératives telles que : chatGpt.

Il est évident, qu'il ne s'agit pas ici de manipuler des réseaux de neurones, mais nous allons dans ce projet nous focaliser sur la méthode basée sur le nombre d'occurrences des mots permettant de générer des réponses intelligentes à partir d'un corpus de textes. L'objectif est de concevoir un système qui peut répondre à des questions en se basant sur la fréquence des mots dans le corpus.

Cette application repose sur l'algorithme de base suivant :

Prétraitement des données : votre programme commence par collecter et prétraiter un ensemble de documents afin de comprendre la nature de leurs contenus avant de s'en servir lors de l'élaboration des réponses. Cette phase permet de nettoyer le texte en supprimant la ponctuation, en convertissant les lettres en minuscules, et en divisant le texte en mots (ou "tokens").

Création d'une matrice TF-IDF : Pour chaque mot unique dans les documents, vous serez amenés à calculer un vecteur TF-IDF en utilisant la méthode TF-IDF. Chaque mot est associé à un vecteur de dimension égale au nombre de documents dans le corpus. Cela crée une matrice TF-IDF où chaque ligne représente un mot et chaque colonne représente un document.

Représentation des questions : Lorsqu'une question est posée, le chatbot effectue le même prétraitement sur la question. Ensuite, il calcule un vecteur TF-IDF pour cette question en utilisant le même vocabulaire que les documents. Le vecteur de la question est de la même dimension que les vecteurs associés aux mots du corpus.

Calcul de la similarité : Le chatbot calcule la similarité entre le vecteur de la question et les vecteurs des mots du corpus à l'aide de la similarité cosinus ou d'une autre mesure de similarité. Cela lui permet de déterminer quels mots du corpus sont les plus similaires à la question.

Sélection de la meilleure réponse : Le chatbot identifie les mots du corpus les plus similaires à la question en fonction de leur score de similarité TF-IDF. Ensuite, il sélectionne la réponse qui contient le plus grand nombre de ces mots similaires.

Fournir la réponse : Le chatbot renvoie la réponse sélectionnée comme réponse à la question posée.

Travail demandé

Pour vous guider dans la réalisation de cette application, nous allons la travailler en 3 parties :

1. **Partie I** : Développement des fonctions de base qui permettent de bien comprendre le contenu des fichiers fournis.
2. **Partie II** : Calcule de la matrice de similarité et la génération de réponses automatiques.

3. **Partie III** : Généralisation de l'application pour couvrir divers thèmes.

Partie I

Base de fichiers

Dans ce projet, 8 fichiers texte sont fournis dans un répertoire nommé « speeches ». Ces fichiers représentent des discours de 6 présidents français lors de leurs investitures.

Le nom de chaque fichier est sous le format suivant : **Nomination_[nom d'un president][numéro].txt** où :

- **[nom d'un president]** : Est l'un des noms suivants : Chirac, Giscrad d'Estaing, Mitterrand, Macron, Sarkozy
- **[numéro]** : Est un champs optionnel représentant un numéro séquentiel. Il figure dans les noms des fichiers où un même président a prononcé plusieurs discours.

Les fonctions de base

Afin de bien analyser le contenu de ces fichier, il est d'abord demandé de développer les fonctions suivantes :

1. Extraire les noms des présidents à partir des noms des fichiers texte fournis ;
2. Associer à chaque président un prénom ;
3. Afficher la liste des noms des présidents (attention aux doublons) ;
4. Convertir les textes des 8 fichiers en minuscules et stocker les contenus dans de nouveaux fichiers. Les nouveaux fichiers doivent être stockés dans un nouveau dossier appelé « cleaned ». Ce dossier doit se situer dans le répertoire principal où se trouve le programme ***main.py*** et au même niveau que le répertoire « speeches »
5. Pour chaque fichier stocké dans le répertoire « cleaned », parcourir son texte et supprimer tout caractère de ponctuation. Le résultat final doit donner un fichier avec des mots séparés par des espaces. Attention, certains caractères comme l'apostrophe (') ou le tiret (-) nécessitent un traitement spécial pour ne pas causer une concaténation de deux mots (exemple : « elle-même » devrait devenir « elle même » et non pas « ellemême »). Les modifications réalisées à cette étape devraient être stockées dans les mêmes fichiers du répertoire « cleaned ».

Notes :

1. Pour réaliser ces fonctions, il est possible d'utiliser les modules prédéfinis abordés en cours mais aussi le module « os »
2. Le code Python utilisé pour parcourir la liste des fichiers d'une extension donnée et dans un répertoire donné est le suivant :

```python
import os

def list_of_files(directory, extension):

    files_names = []
    for filename in os.listdir(directory):
        if filename.endswith(extension):
            files_names.append(filename)
    return files_names
```

```python
# Call of the function
directory = "./speeches"
files_names = list_of_files(directory, "txt")
print_list(files_names)
```

La méthode TF-IDF

Dans le traitement de texte, il est souvent utile de représenter les mots sous forme de vecteurs numériques afin de faciliter la recherche d'information, la classification de documents et/ou l'analyse de texte.
Une des méthodes basée sur la fréquence pour générer des vecteurs numériques pour les mots s'appelle la méthode **TF-IDF** (**T**erm **F**requency-**I**nverse **D**ocument **F**requency). Voici comment elle fonctionne :

TF (Term Frequency - Fréquence du terme) : La première partie du **TF-IDF** mesure à quelle fréquence un terme (un mot dans notre cas) apparaît dans un document spécifique. Plus un terme apparaît fréquemment dans un document, plus son score **TF** est élevé pour ce document ➔ Pour chaque mot dans chacun des fichiers, il est demandé de compter le nombre de fois où ce mot apparaît. Pour ce faire, il est au minimum nécessaire d'avoir une fonction qui calcule le nombre d'occurrence de chaque mot dans un texte.

- Écrire une fonction qui prend en paramètre une chaine de caractères et qui retourne un dictionnaire associant à chaque mot le nombre de fois qu'il apparait dans la chaine de caractères.

IDF (Inverse Document Frequency - Fréquence Inverse du Document) : La deuxième partie du **TF-IDF** mesure l'importance d'un terme dans l'ensemble du corpus de documents. Les termes qui sont très fréquents dans de nombreux documents auront un score **IDF** plus faible, tandis que les termes qui sont rares auront un score **IDF** plus élevé. ➔ Pour chaque mot, calculer le **logarithme décimal** de l'inverse de la proportion de documents dans le corpus qui contiennent ce mot. Cela permet de donner plus de poids aux mots rares et de réduire le poids des mots communs.
Plusieurs formules de calcul du score IDF existent. Par souci de simplification, la formule à utiliser ici est la suivante (sans le +1) :

$$IDF\ (mot) = log_{10}\left(\frac{Nombre\ total\ de\ documents\ dans\ le\ corpus}{Nombre\ de\ documents\ du\ corpus\ contenant\ le\ mot}\right)$$

- Écrire une fonction qui prend en paramètre le répertoire où se trouve l'ensemble des fichiers du corpus et qui retourne un dictionnaire associant à chaque mot son score **IDF**.

Note :

Il est autorisé d'utiliser le module « math » afin de pouvoir appeler sa fonction prédéfinie calculant le logarithme (**log10**).

Score final : Le score **TF-IDF** d'un mot dans un document est obtenu en multipliant le score **TF** par le score **IDF**.

- TF-IDF = TF * IDF.

Ainsi, le score **TF-IDF** d'un mot dans un document donné est un vecteur numérique qui reflète à la fois la fréquence du mot dans ce document et son importance relative par rapport à l'ensemble du corpus.

Pour générer des vecteurs numériques pour les mots, chaque mot dans le corpus de documents se voit attribuer un vecteur de scores **TF-IDF**. Ces vecteurs numériques permettent de représenter la distribution de chaque mot dans tout le corpus. L'ensemble de ces vecteurs forme une matrice appelée "**matrice TF-IDF**", où chaque ligne représente un mot et chaque colonne représente un document.

- Écrire une fonction qui prend en paramètre le répertoire où se trouvent les fichiers à analyser et qui retourne au minimum la matrice TF-IDF.
- Le nombre de lignes de la matrice résultante doit être égal au nombre de mots uniques dans tous les fichiers, et le nombre de colonnes doit être égal au nombre de fichiers se trouvant dans le répertoire

  ➔ Si besoin, écrire une fonction qui calcule la transposée d'une matrice (**Attention !!** ne pas utiliser de fonction prédéfinie)

Exemple d'une partie de la matrice TF-IDF:

```
[0.98, 0.98, 0.0, 0.98, 0.0, 0.0, 0.0, 0.0]
[1.39, 0.0, 1.39, 0.0, 0.0, 0.0, 0.0, 0.0]
[0.47, 0.0, 0.0, 0.47, 0.47, 0.47, 0.47, 0.0]
[1.39, 0.0, 0.0, 1.39, 0.0, 0.0, 0.0, 0.0]
[1.88, 0.47, 0.94, 1.41, 0.47, 0.0, 0.0, 0.0]
[2.08, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[2.08, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[0.69, 0.0, 0.69, 0.69, 0.0, 0.69, 0.0, 0.0]
[0.69, 0.69, 0.0, 2.77, 0.0, 0.0, 0.69, 0.0]
[0.98, 0.98, 0.0, 0.0, 1.96, 0.0, 0.0, 0.0]
[1.39, 0.0, 0.0, 0.0, 1.39, 0.0, 0.0, 0.0]
```

Fonctionnalités à développer

Sur la base des fonctions précédentes, écrire des programmes qui permettent de :

1. Afficher la liste des mots les moins importants dans le corpus de documents. Un mot est dit non important, si son **TD-IDF** = 0 dans tous les fichiers.

   Exemple :

```
['se', 'son', 'a', 'des', 'ce', 'dans', 'j', 'pour', 'je', 'l', 'qui', 'du', 'messieurs', 'mesdames', 'faire', 'de', 'peuple', 'histoire', 'est', 'france', 'les',
 'en', 'qu', 'la', 'aux', 'il', 'que', 'une', 'et', 'par', 'le', 'mais']
```

2. Afficher le(s) mot(s) ayant le score **TD-IDF** le plus élevé
3. Indiquer le(s) mot(s) le(s) plus répété(s) par le président Chirac hormis les mots dits « non importants »
4. Indiquer le(s) nom(s) du (des) président(s) qui a (ont) parlé de la « Nation » et celui qui l'a répété le plus de fois
5. Indiquer le premier président à parler du climat et/ou de l'écologie
6. Hormis les mots dits « non importants », quel(s) est(sont) le(s) mot(s) que tous les présidents ont évoqués.

Programme principal

Il s'agit ici d'écrire un programme principal temporaire dans lequel il est demandé de :

6. Avoir un menu
7. Accéder aux fonctionnalités précédentes selon la demande de l'utilisateur

Ne pas oublier que l'utilisation de Git est obligatoire. Toutes les équipes projet doivent avoir un dépôt sur Github et le code doit être partagé avec tous les membres de l'équipe.

Partie II

Après avoir traité les documents du corpus et généré la matrice TF-IDF, nous allons nous intéresser dans cette partie à la génération automatique de réponses aux questions.

Pour ce faire, nous devons :

- D'abord, calculer le TF-IDF de la question posée ;
- Chercher dans le corpus le(s) document(s) le(s) plus pertinent(s), i.e. celui qui contient le plus de mots similaires aux mots de la questions posée ➔ Il contiendrait probablement plus d'éléments de réponses à la question posée ;
- Générer la réponse sur la base du document sélectionné.

Commençons !

1. **Tokenisation de la Question** :

   Tokeniser la question en mots individuels de la même manière que pour les documents du corpus. Cela implique de diviser la question en mots et de supprimer la ponctuation, supprimer les majuscules, supprimer éventuellement les mots vides et tout autre traitement supplémentaire effectué sur les textes des documents du corpus.

   - Écrire une fonction qui prend en paramètre le texte de la question, et qui retourne la liste des mots qui composent la question.

2. Recherche des mots de la question dans le Corpus :

- Écrire une fonction qui permet d'identifier les termes de la question qui sont également présents dans le corpus de documents. Ignorez les termes absents du corpus car ils n'auront pas de valeurs TF-IDF associées. En d'autres termes, il faut chercher les termes qui constituent l'intersection entre l'ensemble des mots du corpus et l'ensemble des mots de la question.

3. Calcul du vecteur TF-IDF pour les termes de la question :

Dans cette partie, la matrice TF-IDF du corpus doit avoir N lignes et M colonnes, où N = nombre de documents dans le corpus (8 dans notre cas) et M = nombre de mots dans le corpus (1681 dans notre cas).

Ceux qui ont représenté la matrice TF-IDF sous forme d'un dictionnaire, doivent s'adapter pour obtenir un vecteur de dimension M associé à chaque document.

Le vecteur TF-IDF de la question doit être de dimension = M (même nombre de colonnes que la matrice TF-IDF). Il est calculé comme suit :

- Associer à chaque mot de la question un score TF (**T**erm **F**requency) qui est égal à la fréquence de chaque mot dans la question ➔ Mettre un 0 pour les mots du corpus qui ne font pas partie de la question.

  Exemple :

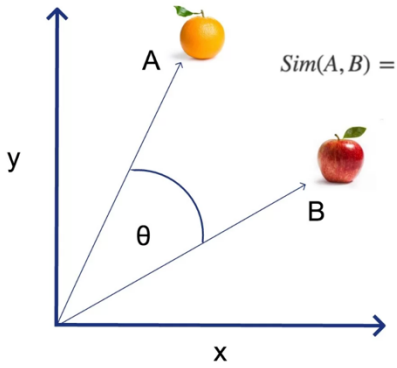
  Si un mot figure deux fois dans une question composée de 10 mots au total, alors le TF de ce mot est de 2/10.
- Les score IDF des mots de la questions sont les mêmes que ceux des mêmes mots dans le corpus. Il suffit donc d'utiliser ces scores IDF préalablement calculés pour calculer le TF-IDF des termes de la question.

  ➔ La fonction python à écrire ici doit retourner le vecteur TF-IDF de la question.

4. Calcul de la similarité :

A présent que le vecteur de la question est générée, il faudra chercher à quel document du corpus il est le plus similaires. Pour cela, nous allons utiliser la similarité de cosinus. C'est une mesure utilisée pour évaluer à quel point deux vecteurs sont similaires dans un espace vectoriel. En particulier, elle est souvent employée dans le traitement du langage naturel pour mesurer la similarité entre deux textes représentés sous forme de vecteurs (par exemple, des vecteurs TF-IDF).

Mathématiquement, la similarité de cosinus permet de mesurer le cosinus de l'angle entre deux vecteurs A et B dans un espace multidimensionnel. Plus l'angle est petit, plus la valeur de son cosinus est élevée et donc la similarité entre les deux vecteurs est élevée.



$$Sim(A,B) = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$$

Dans ce projet, il est donc demandé de calculer la similarité du vecteur de la question avec chacun des N vecteurs (lignes) de la matrice TF-IDF et de désigner le document avec qui la question obtient la plus haute valeur de similarité.

Question	TF-IDF (mot1)	TF-IDF (mot2)	…	…	TF-IDF (motM)

Doc 1	TF-IDF (mot1)	TF-IDF (mot2)	…	…	TF-IDF (motM)
Doc 2	TF-IDF (mot1)	TF-IDF (mot2)	…	…	TF-IDF (motM)
…	…	…	…	…	…
Doc N	TF-IDF (mot1)	TF-IDF (mot2)	…	…	TF-IDF (motM)

Pour calculer la similarité de cosinus entre deux vecteurs A et B sans avoir à manipuler l'angle séparant les deux vecteurs, il faut se baser sur la formule suivante :

$$\text{cosine similarity} = S_C(A,B) := \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}}$$

En Python, cela impose l'implémentation des fonctions suivantes :

a. **Le produit scalaire :**

C'est une fonction qui prend en paramètres deux vecteurs A et B de même dimension M (nombre de mots dans le corpus) et qui permet de calculer et retourner A . B = $\sum_{i=1}^{M} A_i B_i$

b. **La norme d'un vecteur :**

C'est la longueur d'un vecteur. La fonction doit prendre en paramètre un vecteur A puis calcule et retourne la racine carrée de la somme des carrés de ses composantes:

$||A|| = \sqrt{\sum_{i=1}^{M} A_i^2}$

NB : Il est autorisé d'utiliser ici la fonction **sqrt** du package math.

c. Calcul de la similarité :

Il s'agit là d'une fonction qui prend deux vecteurs A et B en paramètres et qui retourne le résultat du score suivant : $\frac{Produit\ scalaire\ de\ A\ et\ B}{Norme\ du\ vecteur\ A\ X\ Norme\ du\ vecteur\ B}$

5. Calcul du document le plus pertinent

Pour avoir le nom du document le plus pertinent, il suffit d'implémenter une fonction qui prend en paramètres la matrice TF-IDF du corpus, le vecteur TF-IDF de la question ainsi que la liste des noms des fichiers dans le corpus. Elle doit calculer la similarité du vecteur de la question avec chacun des vecteurs du document puis retourne le nom du document correspondant à la valeur de similarité la plus élevée.

À ce niveau, le nom du fichier qui sera retourné sera l'un contenu dans le répertoire « *./cleaned* », il faut donc prévoir une fonction qui fournit son équivalent dans le répertoire « *./speeches* »

6. Génération d'une réponse

La génération d'une réponse automatiquement est un processus complexe qui implique l'utilisation de méthodes avancées gérant la sémantique dans le texte. L'objectif ici est de l'effectuer avec la plus simple des méthodes. Néanmoins, toute amélioration apportée à la méthode décrite dans ce document est la bienvenue à condition que celle-ci ne fait pas appel à des fonctions prédéfinies ou des modules de traitement de texte.

La méthode qui est demandée d'implémenter ici est la suivante :

- Dans le vecteur TF-IDF de la question, repérer le mot ayant le score TF-IDF le plus élevé et le retourner.

> **Explication :**
> Un terme avec un TF-IDF important est un terme qui d'un côté a un TF élevé ce qui signifie que le terme apparaît fréquemment dans un texte spécifique par rapport aux autres termes dans ce même texte. Cela suggère une importance potentielle du terme pour comprendre le contenu de ce texte (c'est la question dans notre cas ici). De l'autre côté, cela implique également que ce terme est rare ailleurs dans le corpus. C'est-à-dire que le terme est spécifique à seulement ce texte.
>
> En analyse de texte, un terme avec un TF-IDF élevé peut souvent être considéré comme caractéristique ou informatif pour un texte particulier. Cela peut indiquer des mots-clés, des concepts spécifiques ou des éléments significatifs qui différencient ce texte des autres.

- Dans le document pertinent retourné dans l'étape 5, repérer la première occurrence de ce mot et retourner comme réponse la phrase qui le contient. Une phrase est définie ici comme étant le texte entouré par deux « . ».

Exemple :

Question : `"Peux-tu me dire comment une nation peut-elle prendre soin du climat ?"`

Document pertinent retourné : `investiture_Macron_duplicate.txt`

Mot ayant le TF-IDF le plus élevé : `climat`

La réponse générée : `qu'il s'agisse de la crise migratoire, du défi climatique, des dérives autoritaires, des excès du capitalisme mondial, et bien sûr du terrorisme ; plus rien désormais ne frappe les uns en épargnant les autres.`

7. Affiner une réponse

Pour apporter plus d'âme à la réponse générée et supprimer son aspect brut, il est possible de lui apporter des améliorations comme le fait d'imposer une majuscule en début de phrase et un « . » à la fin.

L'une des suggestions est la mise en place de répliques en fonction de la forme de la question posée. Pour ce faire, établir un dictionnaire de formes de textes possibles auxquelles des modèles de réponses sont associées. Il est également possible d'enrichir ce dictionnaire par des formules de politesse.

```
# Liste de propositions non exhaustives
question_starters = {
    "Comment": "Après analyse, ",
    "Pourquoi": "Car, ",
    "Peux-tu": "Oui, bien sûr!"
}
```

Exemples :

Question 1 : `"Peux-tu me dire comment une nation peut-elle prendre soin du climat ?"`

Réponse 1 : `Oui, bien sûr! Qu'il s'agisse de la crise migratoire, du défi climatique, des dérives autoritaires, des excès du capitalisme mondial, et bien sûr du terrorisme ; plus rien désormais ne frappe les uns en épargnant les autres.`

Question 2 : `"Comment une nation peut-elle prendre soin du climat ?"`

Réponse 2 : `Après analyse, qu'il s'agisse de la crise migratoire, du défi climatique, des dérives autoritaires, des excès du capitalisme mondial, et bien sûr du terrorisme ; plus rien désormais ne frappe les uns en épargnant les autres.`

Programme principal

Le programme principal final doit avoir un menu qui permet de proposer à l'utilisateur deux options :

- Accéder aux fonctionnalités de la partie I selon la demande de l'utilisateur
- Accéder au mode Chatbot permettant à l'utilisateur de poser une question

Partie III : Bonus

Les fonctionnalités décrites dans les parties I et II sont basiques et permettent seulement de décrire une partie du processus complexe d'analyse de texte.

Pour aller plus loin, plusieurs améliorations peuvent être apportées tant qu'elles ne font pas appel aux fonctions prédéfinies, car il ne faut pas oublier que le but premier ici est de démontrer des compétences en analyse de situations et la conception de solutions adéquates:

1. Dans la phase de nettoyage de texte, en plus de la suppression des majuscules et de la ponctuation, il est également possible de supprimer les accents, remplacer des caractères comme « l' » par « le » ou « la » , « qu' » par « qui » ou « que », …
2. Traiter les verbes conjugués
3. Enrichir le corpus par des documents similaires ou traitant des mêmes sujets (climat, nation, …) et observer :
    a. L'impact sur la qualité de la réponse générée
    b. L'impact sur le temps d'exécution du programme
4. Développer l'outil afin qu'il puisse répondre à des questions thématiques. Pour cela :
    a. Penser à charger des fichiers traitant d'autres thèmes. Par exemple, créer une application qui répond aux questions liées à l'un des 17 objectifs (au choix) de l'UNESCO (United Nations Educational, Scientific and Cultural Organization) de développement durable.

    b. Modifier le menu afin de permettre à l'utilisateur de choisir son thème.

Annexe :

Voici quelques références d'où il est possible de télécharger des documents en format texte.

- https://www.gutenberg.org/ Vaste de livres gratuits, domaines publics et œuvres classiques en format texte ;
- https://commoncrawl.org/ Ensemble de données contenant des textes extraits du web ;
- https://www.kaggle.com/datasets ensembles de données pour diverses compétitions et projets, y compris des ensembles de données textuelles ;
- https://dumps.wikimedia.org/ Dumps de Wikipedia disponibles pour des analyses textuelles sur les articles de Wikipedia.