Le dataset **MedQuAD** est est riche en contenu médical structuré et bien adapté pour un chatbot médical. Voici un guide complet pour travailler avec ce dataset depuis le début.

1. Comprendre le dataset MedQuAD

- **MedQuAD** (Medical Question and Answer Dataset) contient des **questions-réponses médicales extraites de sources fiables** comme les National Institutes of Health (NIH).
- Chaque entrée inclut :
  - **Questions** : Questions posées par des patients ou des professionnels.
  - **Réponses** : Réponses détaillées fournies par des sources fiables.
  - **Catégories** : Indiquent le domaine médical, comme diabète, cancer, etc.

2. Télécharger et Préparer les Données

Étape 2.1 : Télécharger le dataset depuis Kaggle

1. Téléchargez le dataset depuis Kaggle : MedQuAD Dataset.
2. Importez le fichier dans **Google Colab** ou votre environnement local :

```
from google.colab import files
files.upload()  # Téléchargez ici le fichier ZIP contenant le dataset
```

5. Décompressez le fichier si nécessaire :

```
!unzip medquad.zip
```

3. Charger et Explorer les Données

Étape 3.1 : Charger les données dans un DataFrame

Le dataset peut être au format JSON ou CSV. Par exemple, s'il est au format CSV :

```
import pandas as pd

# Charger le fichier CSV
df = pd.read_csv("medquad.csv")

# Afficher un aperçu des données
print(df.head())
```

Étape 3.2 : Vérifier les colonnes

Identifiez les colonnes pertinentes, comme :

- **Question** : Les questions posées.
- **Answer** : Les réponses associées.
- **Category** : Le domaine médical de la question.

4. Prétraitement des Données

Étape 4.1 : Nettoyer les questions et réponses

Nettoyez les questions et réponses pour supprimer le bruit comme les balises HTML, les caractères spéciaux, etc. :

```
import re

def nettoyer_texte(texte):
```

```
    # Supprimer les balises HTML
    texte = re.sub(r"<.*?>", "", texte)
    # Supprimer les caractères spéciaux
    texte = re.sub(r"[^a-zA-Z0-9éèêçàùâû\s]", "", texte)
    # Supprimer les espaces multiples
    texte = re.sub(r"\s+", " ", texte)
    return texte.strip()

df["Question_Cleaned"] = df["Question"].apply(nettoyer_texte)
df["Answer_Cleaned"] = df["Answer"].apply(nettoyer_texte)
```

Étape 4.2 : Lemmatisation (optionnel)

Utilisez SpaCy pour appliquer une **lemmatisation** et rendre les textes plus cohérents.

```
import spacy
nlp = spacy.load("en_core_web_sm")

def lemmatiser_texte(texte):
    doc = nlp(texte)
    return " ".join([token.lemma_ for token in doc if not token.is_stop])

df["Question_Lemmatized"] = df["Question_Cleaned"].apply(lemmatiser_texte)
```

5. Modélisation avec TF-IDF

Étape 5.1 : Entraîner TF-IDF

Utilisez **TF-IDF** pour transformer les questions nettoyées en vecteurs numériques.

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Initialiser et entraîner TF-IDF sur les questions nettoyées
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(df["Question_Lemmatized"])
```

Étape 5.2 : Calculer la similarité

Utilisez la **similarité cosinus** pour comparer une question utilisateur avec celles du dataset.

```
from sklearn.metrics.pairwise import cosine_similarity

def trouver_question_similaire(user_question):
    # Nettoyer et vectoriser la question utilisateur
    user_question_cleaned = nettoyer_texte(user_question)
    user_vec = vectorizer.transform([user_question_cleaned])

    # Calculer les similarités
    similarites = cosine_similarity(user_vec, tfidf_matrix)

    # Trouver l'index de la question la plus similaire
    index_similaire = similarites.argsort()[0, -1]

    return index_similaire, similarites[0][index_similaire]
```

6. Amélioration avec BERT

Étape 6.1 : Charger un modèle BERT

Utilisez **BERT** (ou **BioBERT**, optimisé pour les textes médicaux) pour affiner la similarité sémantique.

```
from transformers import AutoTokenizer, AutoModel

tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
model = AutoModel.from_pretrained("bert-base-uncased")

def get_bert_embeddings(text):
    inputs = tokenizer(text, return_tensors="pt", truncation=True,
padding=True, max_length=512)
    with torch.no_grad():
        outputs = model(**inputs)
    return outputs.last_hidden_state.mean(dim=1).squeeze().numpy()
```

Étape 6.2 : Ajouter BERT pour affiner

Ajoutez une phase pour affiner les résultats TF-IDF avec les embeddings BERT :

```
from sklearn.metrics.pairwise import cosine_similarity

def affiner_avec_bert(user_question, index_tfidf):
    # Embedding de la question utilisateur
    user_embedding = get_bert_embeddings(user_question)

    # Embedding de la question similaire trouvée avec TF-IDF
    question_embedding =
get_bert_embeddings(df["Question_Lemmatized"].iloc[index_tfidf])

    # Calculer la similarité cosinus
    return cosine_similarity([user_embedding], [question_embedding])[0][0]
```

7. Intégrer dans Streamlit

Créez une interface Streamlit pour interagir avec votre chatbot.

```
import streamlit as st

st.title("Chatbot Médical - MedQuAD")

user_question = st.text_input("Posez votre question ici :")

if user_question:
    # Trouver la question similaire avec TF-IDF
    index_tfidf, tfidf_score = trouver_question_similaire(user_question)

    # Affiner avec BERT
    bert_score = affiner_avec_bert(user_question, index_tfidf)

    # Afficher les résultats
    st.write("Question similaire :", df["Question"].iloc[index_tfidf])
    st.write("Réponse :", df["Answer"].iloc[index_tfidf])
    st.write("Score TF-IDF :", tfidf_score)
    st.write("Score BERT :", bert_score)
```

8. Tester et Déployer

1. Lancez votre application avec Streamlit :
2. `streamlit run app.py`
3. Si vous êtes sur Colab, utilisez **ngrok** pour exposer l'application en ligne.