NLP PROJECT ONE REPORT

WEB CRAWLER

STEPS ON CREATING THE KNOWLEDGE BASE

STEP 1: IMPORTING RELEVANT LIBRARIES

```
import requests
from bs4 import BeautifulSoup
from nltk.tokenize import word_tokenize
from nltk.corpus import words
```

```
import nltk
# Download the 'words' dataset from NLTK
nltk.download('words')
```

STEP 2: FUNCTION TO FILTER URLS WITH RELEVANT KEYWORDS

```
import re

# Given keywords and compiled patterns
keywords = ['empire', 'kingdom']
keyword_patterns = [re.compile(keyword, re.IGNORECASE) for keyword in keywords]
# Function to check if any keyword matches part of the URL
def check_keywords_in_url(url, patterns):
    for pattern in patterns:
        if pattern.search(url):
            return True  # Return True if any keyword matches
    return False  # Return False if no keyword matches
```

STEP 3: LIST OF URL'S TO SCRAP THROUGH

```
URLS= ['https://history.howstuffworks.com/world-history/10-long-lived-empires.htm','https://www.worldatlas.com/geography/largest-empires-in-history.html','https://ec
       'https://www.businessinsider.com/the-10-greatest-empires-in-history-2011-9#2-the-mongol-empire-was-the-largest-contiguous-empire-the-world-has-ever-seen-9', '
```

STEP 4: SCRAPPING THROUGH THE INITIAL URL'S TO GATHERE MORE RELEVANT URL'S

```
total_links = set()
for url in URLS:
    page = requests.get(url)
    soup = BeautifulSoup(page.text, 'html.parser')
    links = soup.find_all('a')
    external_urls = [link.get('href') for link in links if link.get('href') and (link.get('href').startswith('http://') or link.get('href').startswith('https://'))]
    for external_url in external_urls:
        if check_keywords_in_url(external_url, keyword_patterns):
            if external_url not in total_links:
                total_links.add(external_url)
            if len(total_links) > 100:
                break
```

STEP 5: SCRAPPING TEXT OFF EACH PAGE AND WRITING IT TO TEXT FILE

```
i = 1
for link in total_links:
    page = requests.get(link)
    soup = BeautifulSoup(page.content, 'html.parser')
    text = soup.get_text()
    filename = 'file' + str(i) + '.txt'
    f = open(filename, 'w')
    f.write(text)
    f.close()
    i += 1
```

STEP 6: CLEANING UP THE FILES AND SAVING THE CLEANED-UP TEXT TO AN OUTPUT FILE

```
import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import sent_tokenize

stop_words = set(stopwords.words('english'))
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

# Load set of English words
english_words = set(words.words())

# CLEANING UP THE RAW FILES AND SAVING THEM AS OUTPUT FILES
for i in range(1, len(total_links) + 1):
    filename = 'file' + str(i) + '.txt'
    output_filename = 'output' + str(i) + '.txt'
    f_output = open(output_filename, 'w')
    with open(filename,'r') as f:
        lines = f.read().splitlines()
    for line in lines:
        # line = re.sub(r'[.?!,:;()\-\n\d]',' ', line.lower())
        tokens = word_tokenize(line)
        # Removing stopwords and stemming
        cleaned_tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]
        # removing words which are not in english
        filtered_tokens = ' '.join([word for word in cleaned_tokens if word.lower() in english_words])
        sentences = sent_tokenize(filtered_tokens)
        for sentence in sentences:
            f_output.write(sentence + '\n')
    f_output.close()
```

STEP 7: CREATING A CORPUS FROM CLEANED UP TEXT FILES

```
corpus = []
```

```
# CREATING CORPUS FROM THE CLEANED UP FILES
corpus = []
import re
for i in range(1, 35):
    filename = 'output' + str(i) + '.txt'
    words_to_remove = ['\t', 'th']
    with open(filename, 'r') as f:
        content = f.read().lower()
        pattern = '[^\w\s]'
        content = re.sub(pattern, '', content)
        for word in words_to_remove:
            content = content.replace(word, '')
        corpus.append(content)
```

STEP 8: TOP 25 MOST RELEVANT TERMS FOR KNOWLEDGE BASE AFTER APPLYING TF-IDF AND OTHER TECHNIQUES

```
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np



# Words to remove from the final list
words_to_remove = ['new', 'people', 'share', 'united', 'university', 'ce', 'island', 'world', 'state', 'khan', 'time', 'russia', 'colonial', 'great', 'year'
                   , 'china','known','area', 'million', 'led', 'east', 'land', 'western', 'encyclopedia', 'day', 'term', 'enable', 'early',
                   'moment', 'york', 'article', 'cooky', 'khanate', 'dea', 'son', 'sou', 'tsar', 'established']

# Initialize TF-IDF Vectorizer
vectorizer = TfidfVectorizer(stop_words='english')

# Fit and transform the corpus
X = vectorizer.fit_transform(corpus)

# Sum TF-IDF scores for each term across all documents
sum_tfidf = np.array(X.sum(axis=0)).flatten()

# Get the feature names (words/terms)
words = np.array(vectorizer.get_feature_names())

# Sort the scores
sorted_indices = np.argsort(sum_tfidf)[::-1]

# Initialize an empty list to hold the top words excluding those to be removed
filtered_top_words = []

# Iterate over sorted indices and add words to the filtered list if they are not in the words_to_remove list
for index in sorted_indices:
    if words[index] not in words_to_remove:
        filtered_top_words.append(words[index])
    # Stop once we have the top 25 words after filtering
    if len(filtered_top_words) == 40:
        break

# Display the filtered top 25 words
print(filtered_top_words)
```

```
['empire', 'history', 'war', 'century', 'kingdom', 'dynasty', 'colony', 'advertisement', 'territory', 'king', 'power', 'emperor', 'military', 'trade', 'policy', 'pr
ess', 'rule', 'geography', 'original', 'political', 'imperial', 'government', 'sultan', 'second', 'sea', 'west', 'control', 'end', 'city', 'home', 'country', 'scien
ce', 'reign', 'search', 'independence', 'contact', 'central', 'cultural', 'march', 'sign']
```

STEP 9: BASED ON THE TOP 25 WORDS AND PRIOR KNOWLEDGE BASE, CREATING KNOWLEDGE BASE

```
knowledge_base = [
    'Empire: A sovereign state comprising multiple territories and peoples, ruled by a single supreme authority, often an emperor.\n',
    'History: The study of past events, particularly significant political, cultural, and social movements that have shaped societies.\n',
    'Century: A period of 100 years, often used as a milestone to delineate historical epochs.\n',
    'War: A state of armed conflict between different nations or states or different groups within a nation or state.\n',
    'Colony: A territory under the immediate political control of a state, distinct from the sovereign state, yet still under its dominion.\n',
    'Emperor: The ruler of an empire, commanding vast territories and diverse peoples, often hereditary.\n',
    'Dynasty: A line of hereditary rulers of an empire or kingdom, often characterized by a common surname or royal house.\n',
    'Trade: The exchange of goods and services, crucial for the economic prosperity and interaction of empires with foreign powers.\n',
    'Military: The armed forces of an empire, instrumental in defense, conquest, and maintenance of order within and beyond its borders.\n',
    'Government: The system by which a state or community is governed, often centralized in empires to administer vast territories.\n',
    'Cultural: Pertaining to the arts, customs, traditions, and achievements of a society, enriched and spread by empires.\n',
    'City: A large and significant settlement, often serving as administrative, economic, and cultural hubs in empires.\n',
    'Imperial: Relating to an empire or emperor, denoting authority, or governance over extensive territories.\n',
    'Independence: The condition of a nation, country, or state which exercises self-government, and sovereignty, often achieved through struggle against colonial or
    'Sultan: A title used in Muslim countries for a ruler or nobleman, often denoting sovereign authority similar to an emperor or king.\n'
]
```

STEP 10: PICKLING KNOWLEDGE BASE

```
import pickle
pickle.dump(knowledge_base, open('kb.p', 'wb'))
```

CHATBOT

SYSTEM OVERVIEW

The chatbot system is structured around a main interaction loop where it processes user inputs and generates appropriate responses. It begins by collecting personal information from the user (name, date of birth, likes, dislikes) to create a personalized user model. This model is then used to tailor the chatbot's responses to the individual user. The system uses a knowledge base (loaded from a pickled file) and NLP techniques to interpret the user's queries and produce relevant answers.

NLP Techniques Used

1. **Tokenization**: The system employs NLTK's tokenization methods to break down the text into sentences (**sent_tokenize**) and words (**word_tokenize**). This is crucial for analyzing the structure of the user's input and for the subsequent processing steps, such as vectorization and lemmatization.
2. **Lemmatization**: Using NLTK's **WordNetLemmatizer**, the system converts words into their lemma form. Lemmatization helps in reducing the inflectional forms of words to a common base form, aiding in the normalization of the text for better matching and response generation.
3. **Text Preprocessing**: The chatbot removes punctuation and converts text to lowercase to standardize the input, making it easier to compare user input with the chatbot's knowledge base. This preprocessing step is essential for effective NLP, as it reduces the variability of the text.
4. **TF-IDF Vectorization**: The **TfidfVectorizer** from scikit-learn is used to transform the text data into a matrix of TF-IDF features. TF-IDF (Term Frequency-Inverse Document Frequency) is a numerical statistic that reflects how important a word is to a document in a collection or corpus. This technique is used to weigh the relevance of words within the user's input and the chatbot's knowledge base, facilitating the identification of the most relevant response.
5. **Cosine Similarity**: After vectorizing the text, the system uses cosine similarity to find the most similar response from the knowledge base to the user's query. Cosine similarity measures the cosine of the angle between two non-zero vectors of an inner product space, in this case, the TF-IDF vectors, providing a metric of similarity between the user input and potential responses.

Personalization

The system enhances the user experience through personalization by creating and utilizing a user model that includes the user's name, date of birth, likes, and dislikes. This information allows the chatbot to tailor its responses to each user, making interactions more engaging and relevant.

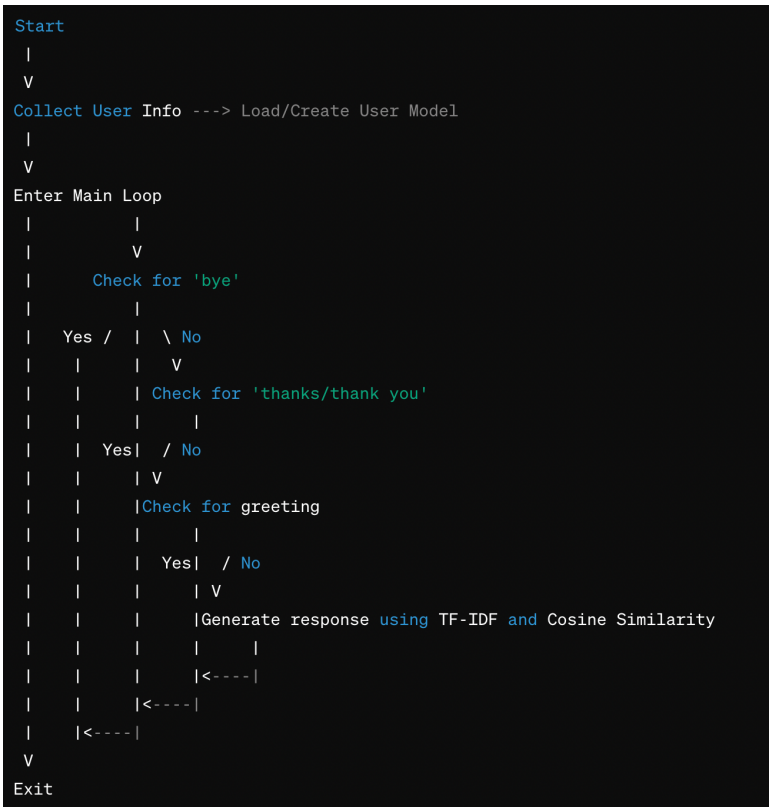
Conclusion

This chatbot system leverages a combination of NLP techniques for text processing, understanding, and response generation. Tokenization, lemmatization, TF-IDF vectorization,

and cosine similarity are core to its ability to interpret user inputs and provide meaningful and personalized responses.

DIALOG TREE OR LOGIC

1. **Initialization and User Model Management**: On startup, the chatbot prompts the user for personal information (name, date of birth, likes, dislikes) and either loads or creates a user model based on this information.
2. **Main Loop**: After initialization, the chatbot enters a main loop where it awaits user input, processes it, and generates responses until the user chooses to exit the conversation.
3. **Input Handling**: Within the main loop, the chatbot first checks for a goodbye message ('bye'), then for gratitude expressions ('thanks', 'thank you'), followed by greetings, and finally any other input.
4. **Response Generation**: For general input (not covered by special cases like greetings or goodbye), the chatbot uses TF-IDF vectorization and cosine similarity to find the most relevant response from the knowledge base, adding personalized content if applicable.



EXAMPLES OF DIALOGUE INTERACTIONS

Appendix A: Knowledge Base Samples

Empire: A sovereign state comprising multiple territories and peoples, ruled by a single supreme authority, often an emperor.

History: The study of past events, particularly significant political, cultural, and social movements that have shaped societies.

Century: A period of 100 years, often used as a milestone to delineate historical epochs.

War: A state of armed conflict between different nations or states or different groups within a nation or state.

Colony: A territory under the immediate political control of a state, distinct from the sovereign state, yet still under its dominion.

Emperor: The ruler of an empire, commanding vast territories and diverse peoples, often hereditary.

Dynasty: A line of hereditary rulers of an empire or kingdom, often characterized by a common surname or royal house.

Trade: The exchange of goods and services, crucial for the economic prosperity and interaction of empires with foreign powers.

Military: The armed forces of an empire, instrumental in defense, conquest, and maintenance of order within and beyond its borders.

Government: The system by which a state or community is governed, often centralized in empires to administer vast territories.

Cultural: Pertaining to the arts, customs, traditions, and achievements of a society, enriched and spread by empires.

City: A large and significant settlement, often serving as administrative, economic, and cultural hubs in empires.

Imperial: Relating to an empire or emperor, denoting authority, or governance over extensive territories.

Independence: The condition of a nation, country, or state which exercises self-government, and sovereignty, often achieved through struggle against colonial or imperial rule.

Sultan: A title used in Muslim countries for a ruler or nobleman, often denoting sovereign authority similar to an emperor or king.

Appendix: Sample User Models

USER 1:

```
{"name": "jimmy", "personal_info": "02/22/1990", "likes": "pizza", "dislikes": "burger"}
```

USER 2:

```
{"name": "nishan", "personal_info": "10/28/1993", "likes": "dogs", "dislikes": "cats"}
```

Evaluations of the chatbot and analysis of its strengths and weaknesses

Strengths:

1. **Personalization**: The chatbot creates a user model based on personal information such as name, likes, and dislikes, which allows for personalized interactions and can enhance user engagement.
2. **Natural Language Understanding**: By employing tokenization, lemmatization, and TF-IDF vectorization, the chatbot demonstrates a fundamental level of natural language understanding, enabling it to process and respond to user queries effectively.
3. **Knowledge-Based Responses**: The chatbot has a solid knowledge base centered on historical, cultural, and political terms, allowing it to deliver informative content in those areas.
4. **Greeting Recognition**: The chatbot can recognize and respond to greetings, which helps in creating a friendly and welcoming interaction environment.
5. **Dynamic Response Generation**: The use of cosine similarity to match user queries with knowledge base entries allows the chatbot to generate responses dynamically, which can be more engaging than static, predefined answers.

Weaknesses:

1. **Contextual Understanding**: The chatbot's ability to understand context and maintain it over multiple turns of conversation is not evident from the code provided. It may not handle complex interactions that require memory of the conversation history.
2. **Scalability**: The chatbot's reliance on a static knowledge base may limit its scalability and the ease with which new information can be integrated.

3. **Error Handling**: The chatbot's response when it does not understand a query is a generic apology. It does not offer follow-up questions or suggestions to guide the user toward more effective interactions.
4. **Limited Domain**: The chatbot's knowledge is limited to the topics present in the static knowledge base, and it may not perform well on topics outside of this domain.

Survey results

Person 1:

1. **Understanding and Relevance**: The chatbot understood my questions and provided relevant answers.
   Agree – 4

2. **Personalization**: The chatbot personalized the conversation based on my likes and dislikes.
   Neutral - 3
3. **User Experience**: Interacting with the chatbot was a pleasant and engaging experience.
   Strongly Agree – 5

Person 2:

1. **Understanding and Relevance**: The chatbot understood my questions and provided relevant answers.
   Agree – 5

2. **Personalization**: The chatbot personalized the conversation based on my likes and dislikes.
   Neutral - 3
3. **User Experience**: Interacting with the chatbot was a pleasant and engaging experience.
   Strongly Agree - 4