Aim: - Implement Information Retrieval for extracting Text from Webpages and Image.

Requirement: -Python versions 3.7, 3.8, 3.9, 3.10 or 3.11, jupyter notebook

Theory: -

To implement information retrieval for extracting text from webpages in NLP, you can follow these general steps:

1. Web Scraping: Use a web scraping library like BeautifulSoup or Scrapy to fetch the HTML content of the webpage.

2. Parsing HTML: Parse the HTML content to extract relevant information. You can use libraries like BeautifulSoup or lxml for this task. Focus on extracting the text from HTML tags such as <p>, <h1>, <h2>, etc., which typically contain the main content of the webpage.

3. Text Preprocessing: Preprocess the extracted text to clean and normalize it. This may involve removing HTML tags, handling special characters, converting to lowercase, removing stopwords, and applying stemming or lemmatization.

4. Text Extraction: Apply text extraction techniques to identify and extract specific information from the preprocessed text. This could involve keyword matching, regular expressions, or more advanced techniques like named entity recognition (NER) or part-of-speech (POS) tagging.

5. Ranking and Filtering: If you want to retrieve specific information or prioritize relevant content, you can apply ranking and filtering techniques. This may include calculating relevance scores based on keyword matching or using machine learning algorithms to classify and rank the extracted text.

6. Indexing: Build an index to efficiently store and retrieve the extracted information. This can be done using data structures like inverted indexes or search engines like Elastic search.

7. Query Processing: Implement a query processing mechanism that takes user queries and retrieves relevant information from the indexed data. This can involve techniques like keyword matching, fuzzy matching, or using advanced search algorithms.

8. User Interface: Develop a user interface or API to interact with the information retrieval system, allowing users to input queries and receive relevant results.

BeautifulSoup4 is a popular Python library used for web scraping and parsing HTML and XML documents. It provides a convenient way to extract data from HTML or XML files by navigating the parse tree and searching for specific elements or attributes.

Some key features of BeautifulSoup4 include:

1. Parsing: BeautifulSoup4 can parse HTML and XML documents and create a parse tree, which represents the structure of the document. It handles imperfect and poorly formatted HTML/XML gracefully.

2. Navigation: It allows you to navigate the parse tree using methods like find(), find_all(), and CSS selectors to locate specific elements based on tags, attributes, or their relationships with other elements.

3. Extraction: BeautifulSoup4 provides methods to extract data from HTML/XML elements. You can retrieve the text content of an element using get_text() or access its attributes using dictionary-like notation.

4. Manipulation: It allows you to modify the parse tree by adding, modifying, or removing elements and attributes. This can be useful when you need to clean or transform the data during the scraping process.

5. Integration: BeautifulSoup4 works well with popular Python libraries such as requests for fetching webpages, and it can handle different encodings and character sets.

Implement Information Retrieval for extracting Text from Webpages.

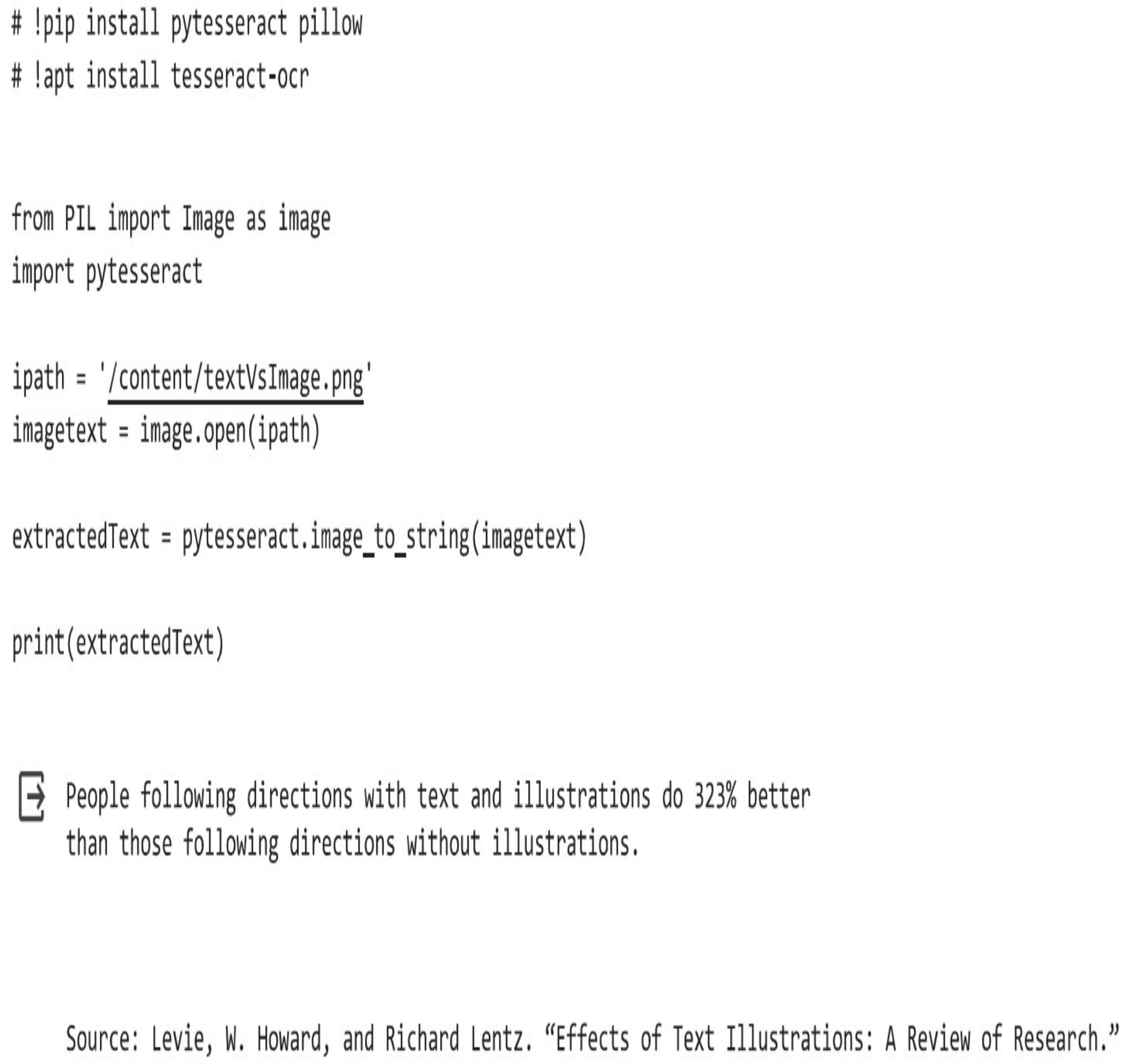
```
!pip install requests beautifulsoup4
```

```
import requests
from bs4 import BeautifulSoup

# Function to extract text from a webpage
def extract_text(url):
    # Send a GET request to the webpage
    response = requests.get(url)

    # Check if the request was successful
    if response.status_code == 200:
        # Parse the HTML content using BeautifulSoup
        soup = BeautifulSoup(response.content, 'html.parser')

        # Extract text from specific HTML tags
        text = ''
        for tag in soup.find_all(['p', 'h1', 'h2']):
            text += tag.get_text() + '\n'

        # Return the extracted text
        return text
    else:
        print('Failed to retrieve the webpage:', response.status_code)
        return None
```

```
# Example usage
url = 'https://amazon.in'  # Replace with the URL of the webpage you want to extract text from
extracted_text = extract_text(url)
if extracted_text:
    print(extracted_text)
```

```
Makeup products
New looks for the new season
Do up your home
Smart gadgets by Amazon
Value bazaar
Work from home essentials
Revamp your home in style
Innovations from Emerging Indian Brands
```

Implement Information Retrieval for extracting Text from Webpages.



```
# !pip install pytesseract pillow
# !apt install tesseract-ocr


from PIL import Image as image
import pytesseract

ipath = '/content/textVsImage.png'
imagetext = image.open(ipath)

extractedText = pytesseract.image_to_string(imagetext)

print(extractedText)
```

```
People following directions with text and illustrations do 323% better
than those following directions without illustrations.



Source: Levie, W. Howard, and Richard Lentz. "Effects of Text Illustrations: A Review of Research."
```