Creating Image Captioning Dataset

Module 1:

The Home Page of Google News is Scrapped using the requests and BeautifulSoup Libraries. The URL of the Home Page is read from a file "config.txt". If the Google News URL changes then the URL in the config file should be replaced by the new URL.

Code:

```
import requests

from bs4 import BeautifulSoup

# To Scrape the Home Page of Google News

# URL is configured by a config file

def Scrape_HomePage(config_file = "config.txt"):

    # Reading the WebPage from a config file

    with open(config_file, 'r') as file:

        urls = [url.strip() for url in file]

    url = urls[0]

    print("WebPage URL:", url)

    print()

    # Scraping the WebPage

    # Sending the request

    req = requests.get(url)

    # Checking the response

    if req.status_code == 200:

        print("Response of the Server for 'get' request to HomePage(Google News) URL:Request Successfull.")

        print()

    else:
```

```
        print(f"Failed to retrieve the page. Status code: {req.status_code}")
        exit()
    # Parsing the response to navigate to find the desired element
    s = BeautifulSoup(req.content, 'html.parser')
    return s
```

Module 2:

To Scrape the Top Stories Link from the Home Page that is scraped. The string "Top stories" is read from the "config.txt" file. If the string changes in the Home Page, then the string should be replaced by the new string in the config file before running the script.

Code:

```
import requests
from bs4 import BeautifulSoup
def Scrape_TopStories(s,config_file = "config.txt"):
    #s = Scrape_HomePage(config_file)
    with open(config_file, 'r') as file:
        urls = [url.strip() for url in file]
    # Reading from the config file.
    heading = urls[1]
    print(heading)
    print()
    # Scraping the Top Stories Link from the Home Page of Google News.
    # To find all <a> tags on the page.
    Links = s.find_all("a")
    # It was found on inspecting the Google news WebPage that the Top stories link is in one of the <a> tags.
    topStory_Link = None
```

```
    #Iterating through all the links and finding if the text in <a> tag matches Top stories read from config file.
    for link in Links:
        if(link.get_text() == heading):
            topStory_Link = link.get('href')
            break
    if topStory_Link:
        topStory_Link = "https://news.google.com"+topStory_Link[1:]
    print("Top Stories URL: ", topStory_Link)
    print()
    #Sending the request
    topStories_req = requests.get(topStory_Link)
    # Checking the response
    if topStories_req.status_code == 200:
        print("Response of the Server for 'get' request to Top Stories URL: Request Successfull.")
        print()
    else:
        print(f"Failed to retrieve the page. Status code: {topStories_req.status_code}")
        exit()
    S = BeautifulSoup(topStories_req.content, 'html.parser')
    return S


# In Module 2, the Server responds with the HTML content(WebPage @ topStory_Link) which is retrieved using the requests library.

# Using BeautifulSoup, the HTML content is parsed to extract specific elements, such as headlines, links, or any other relevant information.

# By directly fetching the page's source code, it helps avoid issues related to lazy loading, where certain elements might only appear after JavaScript execution.
```

requests retrieves the initial HTML content, the structured data can be accessed immediately without waiting for dynamic content to load asynchronously.

Module 3:

To Extract the Thumbnails and the corresponding headlines for all the stories at Top Stories page which is scraped. The Top Stories page is scraped to obtain the full HTML content of the page, avoiding the need to handle lazy loading. Since in lazy loading certain element would appear only after execution of the Java script, By directly fetching the page's source code, it helps avoid issues related to lazy loading. Using BeautifulSoup, the HTML content is parsed to extract specific elements, such as headlines, links, or any other relevant information.

Code:

```
# On Inspecting the Top stories WebPage, it was observed that the images and the corresponding headlines are clubbed within <article> tag.
def Extract_HeadLine_Thumbnail(S):
    # To find all <article> tags on the page.
    articles = S.find_all("article")
    Images = []
    Headlines = []
    # Iterating over each article tag
    for article in articles:
        images = article.find("img")
        img_url = images["src"] if images else "No image"
        if(img_url[:5] == '/api/'):
            Images.append("https://news.google.com"+img_url)
            headline = article.find_all("a")
            Headlines.append(headline[1].get_text())
    Data = [] # (Image_URL, Headline) -> tuples
    for i in range(len(Images)):
        Data.append((Images[i], Headlines[i]))
```

```
    print("Thumbnails and the Headlines are extracted!")

    return Data
```

Module 4:

To store the extracted (Image, Headline) data into the database. PGSQL is the chosen database. A new Database called News is created. A function to create the tables is written, it creates two tables in the News database. One for the Image data and another for the headlines. The headlines and the images are linked using the Image_ID. If a image is removed from the database then the corresponding headline is also removed. Two functions Insert_Image and Insert_headline is created to add Images and headlines to the database respectively. In the Insert_Image function if there is an error in downloading the image, then in the Insert_headline the corresponding headline is skipped from adding to the database. The images are downloaded and stored as a binary file in the database. The original images could be reconstructed by extracting the binary files from the database.

Code:

```
import psycopg2

import requests

import sys

import io

from io import BytesIO

# Set the standard output encoding to UTF-8

sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')

# Database connection details

DB_NAME = "News"

DB_USER = "postgres"

DB_PASSWORD = "manu1609"

DB_HOST = "localhost"

DB_PORT = "5432"
```

```
def create_tables():
    connection = psycopg2.connect(dbname=DB_NAME, user=DB_USER,
password=DB_PASSWORD, host=DB_HOST, port=DB_PORT)
    cur = connection.cursor()
    # Creating images table
    cur.execute("""
        CREATE TABLE IF NOT EXISTS images (
            image_id SERIAL PRIMARY KEY,
            image_data BYTEA NOT NULL
        );
    """)

    # Creating headlines table with only image_id and headline
    cur.execute("""
        CREATE TABLE IF NOT EXISTS headlines (
            headline_id SERIAL PRIMARY KEY,
            headline TEXT UNIQUE NOT NULL,
            image_id INTEGER REFERENCES images(image_id) ON DELETE CASCADE
        );
    """)

    connection.commit()
    cur.close()
    connection.close()
    print("Tables created successfully.")
    print("One Table for the Image data and the other for the corresponding Headline.")
    print()
```

```
def Insert_image(image_url):
    r = requests.get(image_url)
    if r.status_code != 200:
        print(f"Failed to download image: {image_url}")
        return None
    img_data = BytesIO(r.content)   # Image data stored as binary
    connection = psycopg2.connect(dbname=DB_NAME, user=DB_USER,
password=DB_PASSWORD, host=DB_HOST, port=DB_PORT)
    cur = connection.cursor()
    cur.execute("INSERT INTO images (image_data) VALUES (%s) RETURNING image_id;",
(img_data.getvalue(),))
    image_id = cur.fetchone()[0]
    connection.commit()
    cur.close()
    connection.close()
    return image_id


def Insert_headline(headline, image_id):
    # Insert a headline with an associated image.
    if image_id is None:
        print(f"Skipping headline due to missing image: {headline}")
        return
    connection = psycopg2.connect(dbname=DB_NAME, user=DB_USER,
password=DB_PASSWORD, host=DB_HOST, port=DB_PORT)
    cur = connection.cursor()
    cur.execute("INSERT INTO headlines (headline, image_id) VALUES (%s, %s);",
                (headline, image_id))
    connection.commit()
    print(f"Inserted headline: {headline}")
```

```
    cur.close()

    connection.close()
```

Module 5:

It connects to the News database. The check function is written such that when it receives a headline it returns if the headline is present or not in the database. If the headline is not present then the data would be added by invoking the Insert_image and Insert_headline functions.

Code:

```
import psycopg2

DB_NAME = "News"

DB_USER = "postgres"

DB_PASSWORD = "manu1609"

DB_HOST = "localhost"

DB_PORT = "5432"

# To check if the headline is already present in the Database.

# If the headline is unique then only the data(Image, Headline) is added.

def check(headline):

    connection = psycopg2.connect(dbname=DB_NAME, user=DB_USER,
password=DB_PASSWORD, host=DB_HOST, port=DB_PORT)

    cur = connection.cursor()

    cur.execute("SELECT COUNT(*) FROM headlines WHERE headline = %s;", (headline,))

    count = cur.fetchone()[0]

    if(count > 0):

        print("HeadLine already present in DB: ", headline)

    cur.close()

    connection.close()

    return count > 0
```

Module 6:

The module invokes Scrape_HomePage from Module 1, Scrape_TopStories from Module 2, Extract_HeadLine_Thumbnail from Module 3, create_tables, Insert_image, Insert_headline from Module 4 and check from Module 5. It creates a file called, pipeline.log where it logs all the information about running status of each Module along with the time and date. It indicates if the log is a INFO or and an ERROR. The module runs all the invoked functions in a cascaded manner passing the necessary attributes for the subsequent functions. It call the function AddData_to_DB which checks if each headline is present in database or not and then adds the data is its not present in the database.

Code:

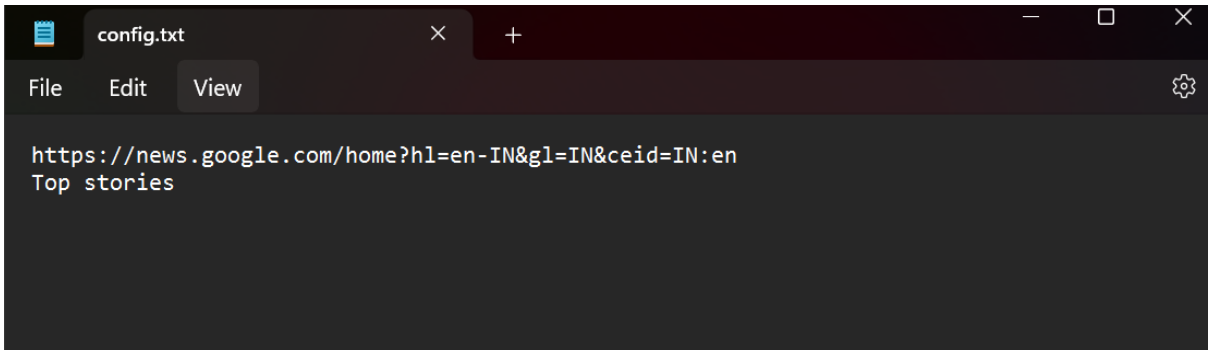
```
import logging

import sys

import time


from Module1 import Scrape_HomePage

from Module2 import Scrape_TopStories

from Module3 import Extract_HeadLine_Thumbnail

from Module4 import create_tables, Insert_image, Insert_headline

from Module5 import check


log_file = "pipeline.log"

logging.basicConfig(

    filename=log_file,

    level=logging.INFO,

    format="%(asctime)s - %(levelname)s - %(message)s",

    datefmt="%Y-%m-%d %H:%M:%S",

)

def log_event(message, level = 'info'):

    if level == "info":
```

```
        logging.info(message)
    elif level == "Error":
        logging.error(message)
    print(message)

def AddData_to_DB(data):
    for image_path, headline in data:
        if not check(headline):
            image_id = Insert_image(image_path)
            Insert_headline(headline, image_id)

def PipeLine():
    start = time.time()
    log_event("PipeLine Execution Started!")
    try:
        log_event("Executing Module 1")
        s = Scrape_HomePage()
        log_event("Completed Executing Module 1")
        log_event("Executing Module 2")
        S = Scrape_TopStories(s)
        log_event("Completed Executing Module 2")
        log_event("Executing Module 3")
        Data = Extract_HeadLine_Thumbnail(S)
        log_event("Completed Executing Module 3")
        log_event("Executing Module 4")
        create_tables()
        log_event("Completed Executing Module 4")
        log_event("Executing Module 5")
```

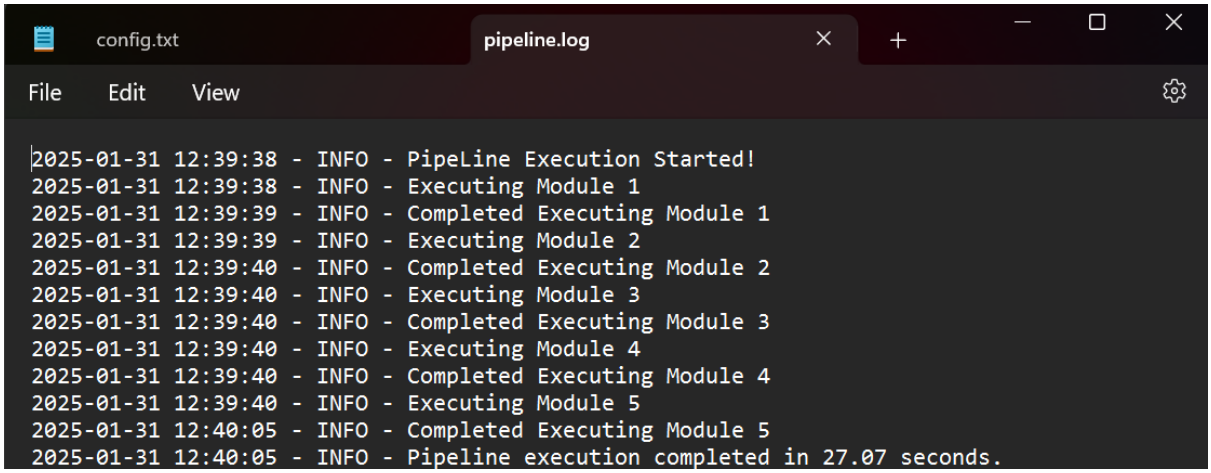
```
        AddData_to_DB(Data)
        log_event("Completed Executing Module 5")
    except Exception as e:
        log_event(f"Pipeline execution failed: {str(e)}", level="error")
        sys.exit(1)
    end = time.time()
    log_event(f"Pipeline execution completed in {end - start:.2f} seconds.")
if __name__ == "__main__":
    PipeLine()
```

All the codes snippet submitted have the necessary explanations in the comments.

Config.txt:



pipline.log:



Submitted by,

Manoj

DA24S018