DA5402: Machine Learning Operations Laboratory

Assignment 1

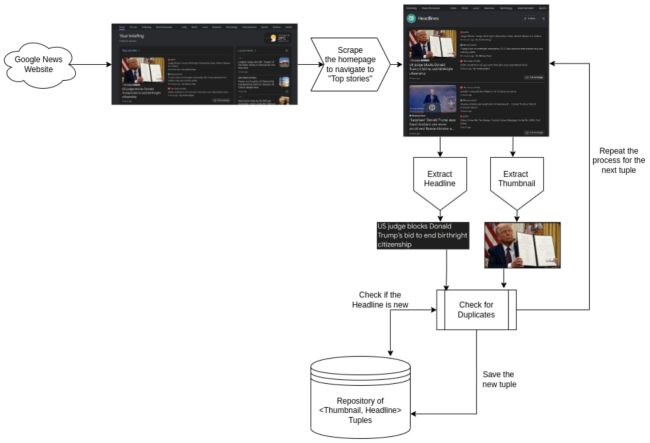
Nandhakishore C S (DA24M011)

February 2, 2025

1 Problem Statement

Flickr30k is a popular open-source image captioning dataset in use today, which you may download from HuggingFace and other archives.

Let's build our own data pipeline to create an "image-captioning" dataset. To get there, we are going to use Google News as the Portkey. Your task is to set up the following pipeline using Python scripts and demonstrate that your pipeline, when set up for automated execution, shall produce a continuous feed of $\langle image, caption \rangle$ tuples.



1.1 Module 1 [5 points]

Create a Python script using any webs crapping libraries to scrape the home page of Google News.Don't hard code the URLs as the home page URLs may change overtime. Ensure that such parameters are configurable via a command line or a configuration file.

Setup

Environment

To scrape a webpage and store it's contents (which are HTML tags and other meta data), we will be using *requests* and *yaml* packages from *Python3* to connect to and from the webpage.

Script

```python
import yaml
import requests

class WebPageScrapper:
    __slots__ = '_config_file'
    def __init__(self, config_file_path: str) -> None:
        if(config_file_path):
            self._config_file = self._load_config(config_file_path)
        else:
            self._config_file = None

```

```
    def _load_config(self, config_file_path: str):
        with open(config_file_path, 'r') as file:
            return yaml.safe_load(file)

    def scrape(self):
        url = self._config_file['webpage']['base_url']
        responce = requests.get(url)
        responce.raise_for_status()
        return responce.text

# -------- Code to test module 1 --------

if __name__ == '__main__':
    website_scrapper = WebPageScrapper(config_file_path = 'config.yaml')
    content = website_scrapper.scrape()
    print(f'Website Content:\n{content}\n')
```

Execution

Do ***python3 module1.py*** in the command line interface to execute the script.

Results

The contents of the scraped webpage will be printed in the terminal.

Observations

The contents of the scrapped webpage are a mix of HTML tags, website data (including port numbers, hashes, submet IDs, IP addresses, etc).

1.2 Module 2 [5 points]

Create a Python script to scrape the "Top Stories" link from the homepage. Don't hard code the "Top stories" string on the home page as the heading may change overtime. Ensure that such parameters are configurable via a command line or a configuration file.

Setup

Environment

- This module requires *bs4* package, which has to installed before hand while running the script. (More on this at the end).
- The *yaml* package only gets the contents from the website which are after the main URL. In this case anything in the URL after "https://news.google.com/".

Script

```
from .module1 import *
from bs4 import BeautifulSoup

class ContentScrapper:
    __slots__ = '_parser', '_config'
    def __init__(self, page_contents: str, config_file: dict) -> None:
        self._parser = BeautifulSoup(page_contents, 'html.parser')
        self._config = config_file

    def find_sub_heading(self, sub_section_str: str = 'sub_heading') -> str:
        pattern = self._config['webpage'][sub_section_str]
        links = self._parser.find_all('a')

        for link in links:
            link_text = link.get_text().strip()
            if(pattern.lower() in link_text.lower()):
                href = link.get('href')
                return self._config['webpage']['base_url'] + href[1:-1]

# -------- Code to test Module2 --------

import yaml
if __name__ == "__main__":
    page_scrapper = WebPageScrapper(config_file_path = 'config.yaml')
    webpage_content = page_scrapper.scrape()

```

```
    with open('config.yaml', 'r') as file:
        config = yaml.safe_load(file)

    topnews_scrapper = ContentScrapper(page_contents = webpage_content, config_file = config)
    top_news_url = topnews_scrapper.find_sub_heading()
    print(f'Top news URL: \n{top_news_url}\n')
```

Execution

Do ***python3 module2.py*** in the command line interface to execute the script.

Results

The URL of the headlines sub section of the google news page will be displayed in the terminal. To cross check the validity of the URL, do cmd + click (or) ctrl + click.

Observations

When the sub heading (i.e.) Top Stories in our problem, when not configured properly, in the .yaml file, the program might give out garbage values.

1.3 Module 3 [10 points]

Create a Python script to extract the thumbnail and the headline of every story from that page. Remember that the page is set up for lazy loading . Your script should do the needful to factor lazy loading. The layout of the "Top stories" page may change overtime, so ensure that you create your module in a way for easy updates later.

Environment

- This module uses *bs4* package, with which the thumbnail (usually image of format .png (or) .webp) is scrapped. Along with the image, the coresponding headline is also taken out.
- To factor out the the lazy loading problem, the script is coded to to wait for five seconds. If the thumbnail or the page is not loaded within the given five second time frame, the script skips that news and movers on the next URL.

Script

```
import requests
import bs4
from bs4 import BeautifulSoup
import time
import json
from datetime import datetime, timedelta

from .module1 import *
from .module2 import *

class ContentExtractor:
    __slots__ = '_config', '_headers'
    def __init__(self, config_file: dict) -> None:
        self._config = config_file
        self._headers = {
            'User-Agent': config_file['webpage']['user_agent']
        }

    def extract_stories(self, url: str, max_retries: int = 2) -> list:
        stories = []
        page = 1

        while(page <= max_retries):
            try:
                response = requests.get(url, headers=self._headers)
                response.raise_for_status()
                soup = BeautifulSoup(response.text, 'html.parser')
                articles = soup.find_all('article')

                for article in articles:
                    story = self._extract_story_data(article)
                    if story and story not in stories:
                        stories.append(story)

                if ((not articles) or (len(articles) == 0)):
                    break

```

```
                page += 1
                time.sleep(2)

            except requests.RequestException as error:
                print(f"Error fetching page {page}: {error}")
                break

        return stories

    def _get_time(self, relative_time: str) -> str:
        try:
            now = datetime.now()

            if 'minute' in relative_time or 'minutes' in relative_time:
                minutes = int(relative_time.split()[0])
                return (now - timedelta(minutes=minutes)).strftime("%Y-%m-%d %H:%M:%S")

            elif 'hour' in relative_time or 'hours' in relative_time:
                hours = int(relative_time.split()[0])
                return (now - timedelta(hours=hours)).strftime("%Y-%m-%d %H:%M:%S")

            elif 'day' in relative_time or 'days' in relative_time:
                days = int(relative_time.split()[0])
                return (now - timedelta(days=days)).strftime("%Y-%m-%d %H:%M:%S")

            elif 'Yesterday' in relative_time:
                return (now - timedelta(days=1)).strftime("%Y-%m-%d %H:%M:%S")

            else:
                return now.strftime("%Y-%m-%d %H:%M:%S")

        except Exception:
            return now.strftime("%Y-%m-%d %H:%M:%S")

    def _extract_story_data(self, article: bs4.element.Tag) -> dict:
        try:
            # Extract headline and URL first
            headline = None
            for a_tag in article.find_all('a'):
                if(a_tag.get_text().strip() is not None):
                    headline = a_tag.get_text().strip()
                    article_url = a_tag.get('href')
                    if((article_url) and (not article_url.startswith('http'))):
                        article_url = f"https://news.google.com{article_url[1:-1]}"
                    else:
                        break

            if(not headline):
                return None

            # Extract thumbnail
            thumbnail_url = None

            # Try figure tag first
            figure = article.find('figure')
            if(figure is not None):
                img = figure.find('img')
                if(img is not None):
                    thumbnail_url = img.get('src') or img.get('data-src')
                    thumbnail_url = f"https://news.google.com{thumbnail_url}"

            # If no image found, skip this story
            if(not thumbnail_url): return

            # Extract and convert publication date
            time_elem = article.find('time')
            relative_time = time_elem.get_text().strip() if time_elem else None
            pub_date = self._get_time(relative_time) if relative_time else datetime.now().strftime("%Y
-%m-%d %H:%M:%S")

            return {
                'Headline': headline,
                'Thumbnail_url': thumbnail_url,
                'Article_url': article_url,
                'Date': pub_date
            }

        except Exception as error:
            print(f"Error:\t {error}")
            return

```

```
    def save_stories(self, content: str, filename: str) -> None:
        filename += '.json'
        with open(filename, 'w', encoding = 'utf-8') as f:
            json.dump(content, f, indent = 2, ensure_ascii = False)

# -------- Code to test Module 3 --------

import yaml
if __name__ == "__main__":

    with open('config.yaml', 'r') as file:
        config_file = yaml.safe_load(file)

    scraper = WebPageScrapper(config_file_path = 'config.yaml')
    homepage_content = scraper.scrape()

    if(homepage_content is not None):
        sub_section_scraper = ContentScrapper(page_contents = homepage_content, config_file =
    config_file)
        sub_section_link = sub_section_scraper.find_sub_heading()

        if(sub_section_link is not None):
            extractor = ContentExtractor(config_file)
            stories = extractor.extract_stories(sub_section_link)
            if stories:
                extractor.save_stories(content = stories, filename='Stories')
                print(f"\nSuccessfully extracted {len(stories)} stories to stories.json\n")
            else:
                print("\nNo stories in the given subsection found\n")
        else:
            print("\nGiven url is invalid\n")
```

Data Format

For the checking the extracted data, the headlie, thumnail and the associated meta data are stored in a .json file. This required .json package to be installed in the computer's python environment.

Execution

Do ***python3 module3.py*** in the command line interface to execute the script.

Results

The script gets the headline + thumbnail of the given page in a .json file. For the overall purpose of the exercise, this is not saved instead. Instead it is directly accessed from the run time memory.

Observations

There are duplicate stories scrapped from the same webpage (as notified in the problem statement)

1.4 Module 4 [10 points]

Create a Python script to store the extracted tuple in a database (pgsql/mariadb/mongodb) are the allowed choice. Your database table should have one table for storing the image data and the other table for storing the headlines and other meta information such as URLs, scrape time stamps, article date, etc.

Setup

- The solutions to this module are coded and tested in a Mac OS based computer. Thus, *PostgreSQL* (database) and *DBeaver* (IDE to handle tables) are installed locally.
- To handle the connection between the scripts (which get the data from the URL using *requests* package) and the database, *psycopg* package is used. It is an adapter to the *PostgreSQL* database.
- To configure the database with the necessary fields, a *.sql* script is run on the *PostgresSQL* server. (see the *.sql* script in the next section).

Environment

The python file for module 4, along with the config file are stored in the same directory with the database server running in the background.

Script(s)

SQL script:

```sql
-- Create the database
CREATE DATABASE news_db;

-- Connect to the database
\c news_db; -- \c is only opplicable for SQL not a general command

-- Create tables
CREATE TABLE news_articles (
    id SERIAL PRIMARY KEY,
    headline TEXT NOT NULL,
    article_url TEXT NOT NULL,
    publication_date TEXT NOT NULL
);

CREATE TABLE news_images (
    id SERIAL PRIMARY KEY,
    article_id INTEGER REFERENCES news_articles(id),
    thumbnail_url TEXT NOT NULL,
    image_data BYTEA NOT NULL
);

-- Create indexes for better query performance
CREATE INDEX idx_publication_date
ON news_articles(publication_date);

CREATE INDEX idx_article_id
ON news_images(article_id);

-- Grant necessary permissions
GRANT ALL PRIVILEGES ON DATABASE news_db TO postgres;
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO postgres;
GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO postgres;
```

Python script:

```python
import requests
import logging
import psycopg2 as psycopg

from .module1 import *
from .module2 import *
from .module3 import *
from .module5 import *

class NewsDB:
    __slots__ = '_connection', '_db_configuration', '_cursor', '_log', '_duplicate_checker'
    def __init__(self, config_file: dict) -> None:
        self._db_configuration = config_file['database']
        self._connection = psycopg.connect(
            dbname      = self._db_configuration['dbname'],
            user        = self._db_configuration['user'],
            password    = self._db_configuration['password'],
            host        = self._db_configuration['host'],
            port        = self._db_configuration['port']
        )
        self._cursor = self._connection.cursor()

        # Setup logging
        logging.basicConfig(
            filename    = 'duplicates.log',
            level       = logging.INFO,
            format      = '%(asctime)s - %(message)s',
            datefmt     = '%Y-%m-%d %H:%M:%S'
        )
        self._log = logging.getLogger('DuplicateChecker')
        self._duplicate_checker = DuplicationChecker(config_file)

    def store_article(self, story: list):
        try:
            # Check for duplicates first
            is_duplicate, reason = self._duplicate_checker.is_duplicate(story)

            if(is_duplicate):
                self._log.info(f"Duplicate found - Headline: {story['Headline']} - {reason}")
                return False

            # Insert article data if not duplicate
            self._cursor.execute(
```

```
                """
                    INSERT INTO news_articles (headline, article_url, publication_date)
                    VALUES (%s, %s, %s)
                    RETURNING id
                """,
                (story['Headline'], story['Article_url'], story['Date'])
            )

            article_id = self._cursor.fetchone()[0]

            # Store image if exists
            if (story['Thumbnail_url'] is not None):
                try:
                    response = requests.get(story['Thumbnail_url'])
                    image_data = response.content

                    self._cursor.execute(
                        """
                            INSERT INTO news_images (article_id, thumbnail_url, image_data)
                            VALUES (%s, %s, %s)
                        """,
                        (article_id, story['Thumbnail_url'], psycopg.Binary(image_data))
                    )
                except Exception as error:
                    self._log.error(f"Error acquiring image: {error}")

            self._connection.commit()
            return True

        except Exception as error:
            self._connection.rollback()
            self._log.error(f"Error storing article: {error}")
            return False

    def close(self):
        try:
            if self._cursor:
                self._cursor.close()
            if self._connection:
                self._connection.close()
            if hasattr(self, 'duplicate_checker'):
                self._duplicate_checker.close()
        except Exception as error:
            self._log.error(f"Error during cleanup: {error}")

# ---------------- Code to test Module 4 ----------------

import yaml
if __name__ == "__main__":
    with open('config.yaml', 'r') as file:
        config = yaml.safe_load(file)

    # Get stories using previous modules
    home_scraper = WebPageScrapper('config.yaml')
    homepage_content = home_scraper.scrape()

    if homepage_content:
        top_stories_scraper = ContentScrapper(homepage_content, config)
        top_stories_url = top_stories_scraper.find_sub_heading()

        if top_stories_url:
            story_extractor = ContentExtractor(config)
            stories = story_extractor.extract_stories(top_stories_url)

            if stories:
                # Store in database
                db = NewsDB(config)
                stored_count = 0
                skipped_count = 0

                for story in stories:
                    if db.store_article(story):
                        stored_count += 1
                    else:
                        skipped_count += 1

                db.close()
                db._log.info(f"Successfully stored {stored_count} articles in database")
                db._log.info(f"Skipped {skipped_count} articles")
            else:
                logging.info("No stories were found")
```

```
        else:
            logging.info("Could not find top stories link")
```

Database

- Now that the *create_table.sql* has created the table, the table should look like (db_ss):

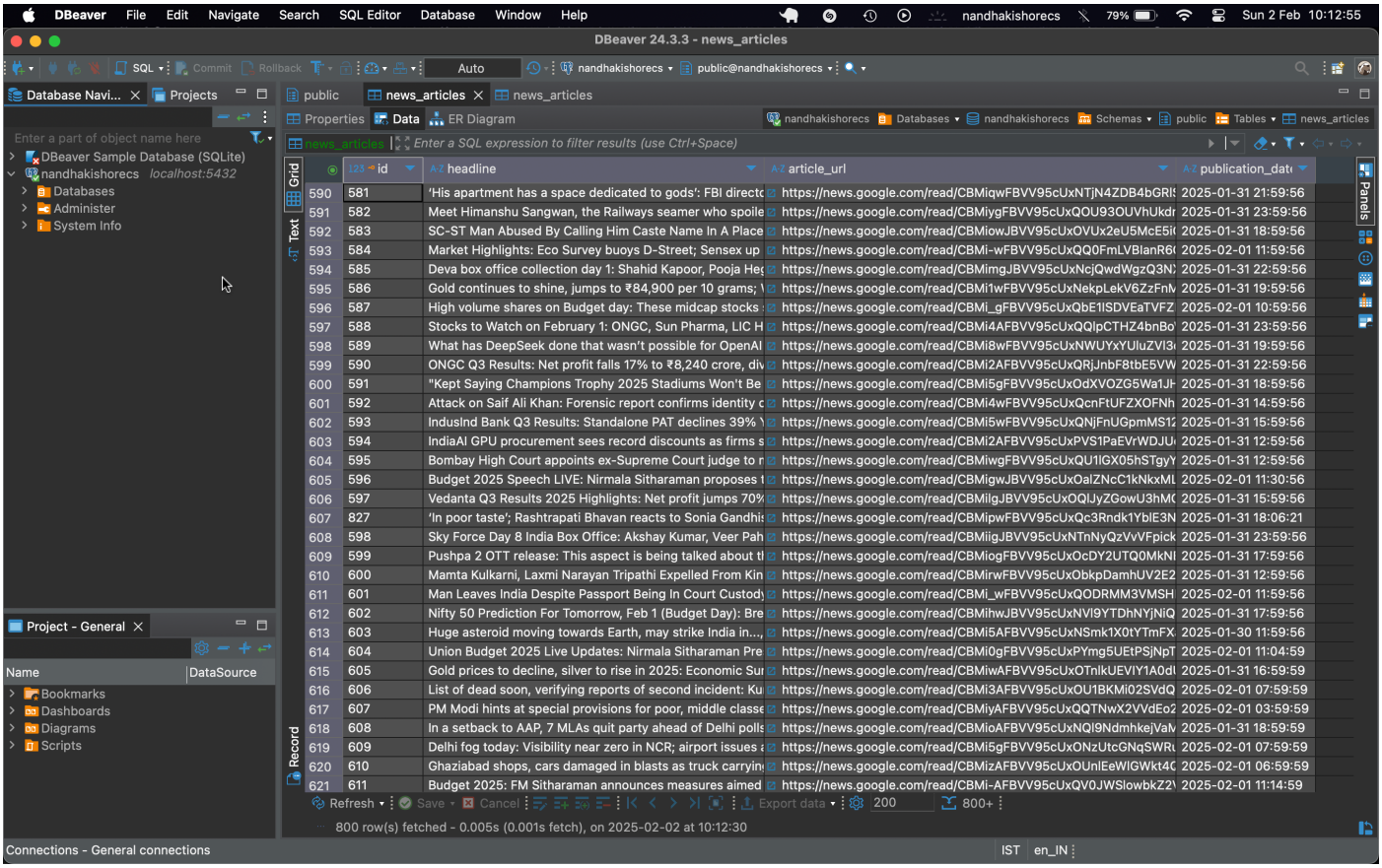


Figure 1: Screenshot of the database the values stored for module 4.

Execution

- To execute the *.sql* script, copy paste the script in the PostgreSQL terminal and press return / enter.
- Do ***python3 module4.py*** in the command line interface to execute the script.

Results

The script gets the values stored in the *PostgresSQL* database in two tables - one with the images and the other with the URLs and it's meta data.

1.5 Module 5 [10 points]

Create a Python script to check if a tuple is already present in the DB based on some de-duplication constraint. One naive constraint is to check the headline, but that's not the best, obviously. Get creative here.

Setup

To Check for duplicates, the script uses the time frame of the URLs and also the module *diff* which uses LCS - Least Common Subsequence to pick out the duplicate / redundant news URLs.

Script

```python
import psycopg2 as psycopg
import logging
from difflib import SequenceMatcher

class DuplicationChecker:
    __slots__ = '_connection', '_db_configuration', '_cursor', '_log'
    def __init__(self, config_file: dict) -> None:
        self._db_configuration = config_file['database']
        self._connection = psycopg.connect(
            dbname      = self._db_configuration['dbname'],
            user        = self._db_configuration['user'],
            password    = self._db_configuration['password'],
            host        = self._db_configuration['host'],
            port        = self._db_configuration['port']
        )
        self._cursor = self._connection.cursor()
        self._log = logging.getLogger('DuplicateChecker')

    def check_similarity(self, headline1: list, headline2: list) -> float:
        return SequenceMatcher(None, headline1.lower(), headline2.lower()).ratio()

    def is_duplicate(self, story: list, same_day_threshold: float = 0.90, different_day_threshold: float = 0.95):
        try:
            story_date = story['Date'][:10]  # Extract YYYY-MM-DD

            # First check same day articles with lower threshold
            self._cursor.execute(
                """
                    SELECT publication_date, headline
                    FROM news_articles
                    WHERE SUBSTRING(publication_date, 1, 10) = %s
                """,
                (story_date,)
            )

            same_day_stories = self._cursor.fetchall()

            for existing_headline, _ in same_day_stories:
                similarity = self.check_similarity(story['Headline'], existing_headline)
                if(similarity >= same_day_threshold):
                    self._log.info(f"Duplicate found - Headline: {story['Headline']} - Same day duplicate found - Similarity: {similarity:.2f}")
                    return True, f"Same day duplicate found - Similarity: {similarity:.2f}"

            # Then check other days with higher threshold
            self._cursor.execute("""
                    SELECT headline, publication_date
                    FROM news_articles
                    WHERE SUBSTRING(publication_date, 1, 10) != %s
                """,
                (story_date,)
            )

            different_day_stories = self._cursor.fetchall()

            for existing_headline, _ in different_day_stories:
                similarity = self.check_similarity(story['Headline'], existing_headline)
                if( similarity >= different_day_threshold):
                    self._log.info(f"Duplicate found - Headline: {story['Headline']} - Different day duplicate found - Similarity: {similarity:.2f}")
                    return True, f"Different day duplicate found - Similarity: {similarity:.2f}"

            return False, "No duplicate found"

        except Exception as error:
            self._log.error(f"Error checking duplication: {error}")
            return False, f"Error: {str(error)}"

    def close(self):
        self._cursor.close()
        self._connection.close()

# -------- Code to test Module 5 --------

import yaml
if __name__ == "__main__":
    with open('config.yaml', 'r') as file:
        config = yaml.safe_load(file)

    checker = DuplicationChecker(config)
```

```
    test_story ={
        "Headline": "Silicon Valley consortium values London Spirit at  295  million in Hundred coup",
        "Thumbnail_url": "https://news.google.com/api/attachments/
    CC8iL0NnNUpVemhzVWpoRVRTMXZUVjlLVFJDZkF4ampCU2dLTWdrQlVJeEJ2YVV0a2dF=-w280-h168-p-df",
        "Article_url": "https://news.google.com/read/
    CBMivAFBVV95cUxNSWV1TWdWNThDZnYxbE8wdDA5NXZoZHdIX1FoT3J2SXFlc2R6ZUdFRjRlUTlGcUdFMmVBTTE4Yk5LMjlORUhYUkExWX
    ?hl=en-IN&gl=IN&ceid=IN%3Ae",
        "Date": "2025-02-01 01:52:08"
    }

    is_duplicate, reason = checker.is_duplicate(test_story)
    print(f"Reason: {reason}")

    checker.close()
```

Environment

This script uses *psycopg* module to establish the connection between the *Python* runtime and the database. When the script is executed, the database server should be on.

Execution

Do ***python3 module5.py*** in the command line interface to execute the script.

Results

The script find outs the duplicate values and the duplicate values are not stored in the database.

1.6 Module 6[10 points]

Write a Python script to orchestrate all the above modules to execute in a cascaded style. The orchestration script should log the time stamps of invocation, error statuses, and other relevant details for debugging later. Moreover,the entire pipeline should be friendly for setting it up as a CronJob [Archwiki].

Environment

To schedule the scrapping for every 30 minutes, the script utilities Cron scheduler to repeat the process once started. It logs the errors and exceptions using the *logging* library in *Python*

Script

```
import logging
from datetime import datetime
import traceback

from .module1 import *
from .module2 import *
from .module3 import *
from .module4 import *

def setup_logging() -> None:
    logging.basicConfig(
        filename    = 'pipeline.log',
        level       = logging.INFO,
        format      = '%(asctime)s - %(levelname)s - %(message)s',
        datefmt     = '%Y-%m-%d %H:%M:%S'
    )
    console_handler = logging.StreamHandler()
    console_handler.setLevel(logging.INFO)
    logging.getLogger().addHandler(console_handler)

def run_pipeline(config):
    try:
        start_time = datetime.now()
        logging.info("\nStarting news scrapping pipeline...\n")

        # Module 1: Scrape homepage
        home_scraper = WebPageScrapper('config.yaml')
        homepage_content = home_scraper.scrape()
        if(not homepage_content):
            raise Exception("\nFailed to scrape webpage\n")

        # Module 2: Extract top stories link
        top_stories_scraper = ContentScrapper(homepage_content, config)
```

```
        top_stories_url = top_stories_scraper.find_sub_heading()
        if not top_stories_url:
            raise Exception("\nFailed to find top stories url\n")

        # Module 3: Extract stories
        story_extractor = ContentExtractor(config)
        stories = story_extractor.extract_stories(top_stories_url)
        if not stories:
            raise Exception("\nNo stories were extracted\n")

        # Module 4: Store in database
        db = NewsDB(config)
        stored_count = 0
        skipped_count = 0

        for story in stories:
            if db.store_article(story):
                stored_count += 1
            else:
                skipped_count += 1

        db.close()

        end_time = datetime.now()
        duration = (end_time - start_time).total_seconds()
        logging.info(f"\nPipeline completed. Stored: {stored_count}, Skipped: {skipped_count},
    Duration: {duration:.2f}s\n")
        return True

    except Exception as error:
        logging.error(f"\nPipeline failed:\t{str(error)}\n")
        logging.error(traceback.format_exc())
        return False

# -------- Code to test Module 6 --------

import yaml
import time

if __name__ == "__main__":
    setup_logging()
    try:
        with open('config.yaml', 'r') as file:
            config = yaml.safe_load(file)

        frequency = config.get('pipeline', {}).get('frequency', )  # Default: .5 hour in seconds

        while True:
            run_pipeline(config)
            logging.info(f"Waiting {frequency} seconds until next run...")
            time.sleep(frequency)

    except KeyboardInterrupt:
        logging.info("Pipeline stopped by user")
    except Exception as error:
        logging.error(f"Fatal error: {str(error)}")
        sys.exit(1)
```

Check the **Overall Execution** section for the details for running *Cron* scheduler.

2 Overall Execution

Setup

- This exercise is done in a Mac OS based computer in a *Python3* virtual environment. Do the following commands in the same order to get the module running.
- The entire scheduler is written as a *Python3* module, where the directory named **DA24M011_Assignment1** itself is a *Python3* module.
- The simplified directory structure of the module looks as following figure. The file **__init__.py** is a empty python script to initialise the module and **__main__.py** is the execution file, which is run when the module is run.

```
.
          DA24M011_Assignment1
                  __init__.py
                  __main__.py
                  __pycache__
                  config.yaml
```

```
                duplicates.log
                module1.py
                module2.py
                module3.py
                module4.py
                module5.py
                module6.py
                pipeline.log
                stories.json
        DA5402_MLOPS_A1.pdf
        create_table.sql
        directory_structure.txt
        mlops_a1
                bin
                include
                lib
                pyvenv.cfg
        pipeline.log
        requirements.txt
```

Environment

- Create a **config.yaml** file in the same directory as the module and fill up the parameters for the user-agent(browser), database and the time after which the scheduler has to repeat the task. The configuration file should look like this.

```yaml
# Base configuration for Google News Scraper
webpage:
  base_url: "https://news.google.com"
  user_agent: "Mozilla/5.0 AppleWebKit/537.36"
  sub_heading: "Top stories"

database:
  dbname: "nandhakishorecs"
  user: "nandhakishorecs"   #replace with your computer's username
  password: ""              #replace with your computer's password
  host: "localhost"
  port: 5432

pipeline:
  frequency: 1800  # Run every half an hour (in seconds)
```

- Create a virtual environment (named as mlops_a1): **$ python3 -m venv mlops_a1**
- Activate the virtual environment: **$ source /path/mlops_a1/bin/activate**
- Install the required packages: **$ pip install -r requirements.txt**
- Run the **create_table.sql** in the **PostgreSQL** server terminal.
- To run the module: **$ python3 -m DA24M011_Assignment.py**
- To stop the program do **ctrl + C**
- To test the modules separately, do **$ python3 module**$< no >$**.py** in the terminal.

Database

- Once the virtual environment is setup, start the PostgreSQL server, terminal and DBeaver.
- Run **create_table.sql** before running the module.

Results

- When executed the module with scrap the "Google News" web page's "Top Stories" section and store the thumbnails and headlines in a database for every 30 minutes until it is stopped by the user.
- Two files will be created: **pipeline.log** and **stories.json**. The log file have the errors and information from the terminal and the json file will have a snapshot of the stored images and headline from the runtime. The json file will only be generated when module 3 is tested separately. It will not be created when the module is executed as a whole.