

# DA5402: Machine Learning Operations Laboratory

## Assignment 1

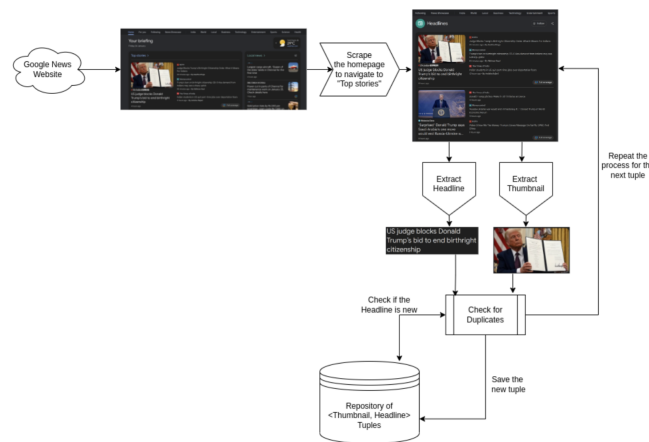
Nandhakishore C S (DA24M011)

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

```
1 import yaml
2 import requests
3
4 class WebPageScrapper:
5     __slots__ = '_config_file'
6     def __init__(self, config_file_path: str) -> None:
7         if(config_file_path):
8             self._config_file = self._load_config(config_file_path)
9         else:
10             self._config_file = None
11
```

```

12 def _load_config(self, config_file_path: str):
13     with open(config_file_path, 'r') as file:
14         return yaml.safe_load(file)
15
16 def scrape(self):
17     url = self._config_file['webpage']['base_url']
18     response = requests.get(url)
19     response.raise_for_status()
20     return response.text
21
22 # ----- Code to test module 1 -----
23
24 if __name__ == '__main__':
25     website_scrapper = WebPageScrapper(config_file_path = 'config.yaml')
26     content = website_scrapper.scrape()
27     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, subnet 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

```

1 from .module1 import *
2 from bs4 import BeautifulSoup
3
4 class ContentScrapper:
5     __slots__ = '_parser', '_config'
6     def __init__(self, page_contents: str, config_file: dict) -> None:
7         self._parser = BeautifulSoup(page_contents, 'html.parser')
8         self._config = config_file
9
10    def find_sub_heading(self, sub_section_str: str = 'sub_heading') -> str:
11        pattern = self._config['webpage'][sub_section_str]
12        links = self._parser.find_all('a')
13
14        for link in links:
15            link_text = link.get_text().strip()
16            if pattern.lower() in link_text.lower():
17                href = link.get('href')
18                return self._config['webpage']['base_url'] + href[1:-1]
19
20 # ----- Code to test Module2 -----
21
22 import yaml
23 if __name__ == "__main__":
24     page_scrapper = WebPageScrapper(config_file_path = 'config.yaml')
25     webpage_content = page_scrapper.scrape()
26

```

```

27 with open('config.yaml', 'r') as file:
28     config = yaml.safe_load(file)
29
30 topnews_scrapper = ContentScrapper(page_contents = webpage_content, config_file = config)
31 top_news_url = topnews_scrapper.find_sub_heading()
32 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 corresponding 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

```

1 import requests
2 import bs4
3 from bs4 import BeautifulSoup
4 import time
5 import json
6 from datetime import datetime, timedelta
7
8 from .module1 import *
9 from .module2 import *
10
11 class ContentExtractor:
12     __slots__ = '_config', '_headers'
13     def __init__(self, config_file: dict) -> None:
14         self._config = config_file
15         self._headers = {
16             'User-Agent': config_file['webpage']['user_agent']
17         }
18
19     def extract_stories(self, url: str, max_retries: int = 2) -> list:
20         stories = []
21         page = 1
22
23         while(page <= max_retries):
24             try:
25                 response = requests.get(url, headers=self._headers)
26                 response.raise_for_status()
27                 soup = BeautifulSoup(response.text, 'html.parser')
28                 articles = soup.find_all('article')
29
30                 for article in articles:
31                     story = self._extract_story_data(article)
32                     if story and story not in stories:
33                         stories.append(story)
34
35                 if ((not articles) or (len(articles) == 0)):
36                     break
37

```

```

38         page += 1
39         time.sleep(2)
40
41     except requests.RequestException as error:
42         print(f"Error fetching page {page}: {error}")
43         break
44
45     return stories
46
47 def _get_time(self, relative_time: str) -> str:
48     try:
49         now = datetime.now()
50
51         if 'minute' in relative_time or 'minutes' in relative_time:
52             minutes = int(relative_time.split()[0])
53             return (now - timedelta(minutes=minutes)).strftime("%Y-%m-%d %H:%M:%S")
54
55         elif 'hour' in relative_time or 'hours' in relative_time:
56             hours = int(relative_time.split()[0])
57             return (now - timedelta(hours=hours)).strftime("%Y-%m-%d %H:%M:%S")
58
59         elif 'day' in relative_time or 'days' in relative_time:
60             days = int(relative_time.split()[0])
61             return (now - timedelta(days=days)).strftime("%Y-%m-%d %H:%M:%S")
62
63         elif 'Yesterday' in relative_time:
64             return (now - timedelta(days=1)).strftime("%Y-%m-%d %H:%M:%S")
65
66         else:
67             return now.strftime("%Y-%m-%d %H:%M:%S")
68
69     except Exception:
70         return now.strftime("%Y-%m-%d %H:%M:%S")
71
72 def _extract_story_data(self, article: bs4.element.Tag) -> dict:
73     try:
74         # Extract headline and URL first
75         headline = None
76         for a_tag in article.find_all('a'):
77             if(a_tag.get_text().strip() is not None):
78                 headline = a_tag.get_text().strip()
79                 article_url = a_tag.get('href')
80                 if((article_url) and (not article_url.startswith('http'))):
81                     article_url = f"https://news.google.com{article_url[1:-1]}"
82             else:
83                 break
84
85         if(not headline):
86             return None
87
88         # Extract thumbnail
89         thumbnail_url = None
90
91         # Try figure tag first
92         figure = article.find('figure')
93         if(figure is not None):
94             img = figure.find('img')
95             if(img is not None):
96                 thumbnail_url = img.get('src') or img.get('data-src')
97                 thumbnail_url = f"https://news.google.com{thumbnail_url}"
98
99         # If no image found, skip this story
100        if(not thumbnail_url): return
101
102        # Extract and convert publication date
103        time_elem = article.find('time')
104        relative_time = time_elem.get_text().strip() if time_elem else None
105        pub_date = self._get_time(relative_time) if relative_time else datetime.now().strftime("%Y-%m-%d %H:%M:%S")
106
107        return {
108            'Headline': headline,
109            'Thumbnail_url': thumbnail_url,
110            'Article_url': article_url,
111            'Date': pub_date
112        }
113
114    except Exception as error:
115        print(f"Error:\t {error}")
116        return
117

```

```

118     def save_stories(self, content: str, filename: str) -> None:
119         filename += '.json'
120         with open(filename, 'w', encoding = 'utf-8') as f:
121             json.dump(content, f, indent = 2, ensure_ascii = False)
122
123 # ----- Code to test Module 3 -----
124
125 import yaml
126 if __name__ == "__main__":
127
128     with open('config.yaml', 'r') as file:
129         config_file = yaml.safe_load(file)
130
131     scraper = WebPageScraper(config_file_path = 'config.yaml')
132     homepage_content = scraper.scrape()
133
134     if(homepage_content is not None):
135         sub_section_scraper = ContentScraper(page_contents = homepage_content, config_file =
136         config_file)
137         sub_section_link = sub_section_scraper.find_sub_heading()
138
139         if(sub_section_link is not None):
140             extractor = ContentExtractor(config_file)
141             stories = extractor.extract_stories(sub_section_link)
142             if stories:
143                 extractor.save_stories(content = stories, filename='Stories')
144                 print(f"\nSuccessfully extracted {len(stories)} stories to stories.json\n")
145             else:
146                 print("\nNo stories in the given subsection found\n")
147         else:
148             print("\nGiven url is invalid\n")

```

## Data Format

For the checking the extracted data, the headline, thumbnail 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 *PostgreSQL* 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:

```
1 -- Create the database
2 CREATE DATABASE news_db;
3
4 -- Connect to the database
5 \c news_db; -- \c is only applicable for SQL not a general command
6
7 -- Create tables
8 CREATE TABLE news_articles (
9     id SERIAL PRIMARY KEY,
10    headline TEXT NOT NULL,
11    article_url TEXT NOT NULL,
12    publication_date TEXT NOT NULL
13 );
14
15 CREATE TABLE news_images (
16     id SERIAL PRIMARY KEY,
17     article_id INTEGER REFERENCES news_articles(id),
18     thumbnail_url TEXT NOT NULL,
19     image_data BYTEA NOT NULL
20 );
21
22 -- Create indexes for better query performance
23 CREATE INDEX idx_publication_date
24 ON news_articles(publication_date);
25
26 CREATE INDEX idx_article_id
27 ON news_images(article_id);
28
29 -- Grant necessary permissions
30 GRANT ALL PRIVILEGES ON DATABASE news_db TO postgres;
31 GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO postgres;
32 GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO postgres;
```

Python script:

```
1 import requests
2 import logging
3 import psycopg2 as psycopg
4
5 from .module1 import *
6 from .module2 import *
7 from .module3 import *
8 from .module5 import *
9
10 class NewsDB:
11     __slots__ = '_connection', '_db_configuration', '_cursor', '_log', '_duplicate_checker'
12     def __init__(self, config_file: dict) -> None:
13         self._db_configuration = config_file['database']
14         self._connection = psycopg.connect(
15             dbname = self._db_configuration['dbname'],
16             user = self._db_configuration['user'],
17             password = self._db_configuration['password'],
18             host = self._db_configuration['host'],
19             port = self._db_configuration['port']
20         )
21         self._cursor = self._connection.cursor()
22
23         # Setup logging
24         logging.basicConfig(
25             filename = 'duplicates.log',
26             level = logging.INFO,
27             format = '%(asctime)s - %(message)s',
28             datefmt = '%Y-%m-%d %H:%M:%S'
29         )
30         self._log = logging.getLogger('DuplicateChecker')
31         self._duplicate_checker = DuplicationChecker(config_file)
32
33     def store_article(self, story: list):
34         try:
35             # Check for duplicates first
36             is_duplicate, reason = self._duplicate_checker.is_duplicate(story)
37
38             if(is_duplicate):
39                 self._log.info(f"Duplicate found - Headline: {story['Headline']} - {reason}")
40                 return False
41
42             # Insert article data if not duplicate
43             self._cursor.execute(
```

```

44         """
45         INSERT INTO news_articles (headline, article_url, publication_date)
46         VALUES (%s, %s, %s)
47         RETURNING id
48         """
49         (story['Headline'], story['Article_url'], story['Date'])
50     )
51
52     article_id = self._cursor.fetchone()[0]
53
54     # Store image if exists
55     if (story['Thumbnail_url'] is not None):
56         try:
57             response = requests.get(story['Thumbnail_url'])
58             image_data = response.content
59
60             self._cursor.execute(
61                 """
62                 INSERT INTO news_images (article_id, thumbnail_url, image_data)
63                 VALUES (%s, %s, %s)
64                 """
65                 (article_id, story['Thumbnail_url'], psycopg.Binary(image_data))
66             )
67         except Exception as error:
68             self._log.error(f"Error acquiring image: {error}")
69
70     self._connection.commit()
71     return True
72
73 except Exception as error:
74     self._connection.rollback()
75     self._log.error(f"Error storing article: {error}")
76     return False
77
78 def close(self):
79     try:
80         if self._cursor:
81             self._cursor.close()
82         if self._connection:
83             self._connection.close()
84         if hasattr(self, 'duplicate_checker'):
85             self._duplicate_checker.close()
86     except Exception as error:
87         self._log.error(f"Error during cleanup: {error}")
88
89 # ----- Code to test Module 4 -----
90
91 import yaml
92 if __name__ == "__main__":
93     with open('config.yaml', 'r') as file:
94         config = yaml.safe_load(file)
95
96     # Get stories using previous modules
97     home_scraper = WebPageScraper('config.yaml')
98     homepage_content = home_scraper.scrape()
99
100     if homepage_content:
101         top_stories_scraper = ContentScraper(homepage_content, config)
102         top_stories_url = top_stories_scraper.find_sub_heading()
103
104         if top_stories_url:
105             story_extractor = ContentExtractor(config)
106             stories = story_extractor.extract_stories(top_stories_url)
107
108             if stories:
109                 # Store in database
110                 db = NewsDB(config)
111                 stored_count = 0
112                 skipped_count = 0
113
114                 for story in stories:
115                     if db.store_article(story):
116                         stored_count += 1
117                     else:
118                         skipped_count += 1
119
120                 db.close()
121                 db._log.info(f"Successfully stored {stored_count} articles in database")
122                 db._log.info(f"Skipped {skipped_count} articles")
123             else:
124                 logging.info("No stories were found")

```

```

125     else:
126         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):

id	headline	article_url	publication_date
590	'His apartment has a space dedicated to gods': FBI direct	https://news.google.com/read/CBMiqwFBVV95cUxNTJN4ZDB4bGRk	2025-01-31 21:59:56
591	Meet Himanshu Sangwan, the Railways seamer who spoile	https://news.google.com/read/CBMiygFBVV95cUxQU093OUVhUkdr	2025-01-31 23:59:56
592	SC-ST Man Abused By Calling Him Caste Name In A Place	https://news.google.com/read/CBMiowJBVV95cUxOVUx2eU5McE5k	2025-01-31 18:59:56
593	Market Highlights: Eco Survey buoys D-Street; Sensex up	https://news.google.com/read/CBMi-wFBVV95cUxQQOfmLVBlanR6i	2025-02-01 11:59:56
594	Deva box office collection day 1: Shahid Kapoor, Pooja He	https://news.google.com/read/CBMimgJBVV95cUxNcQwdWgZQ3N;	2025-01-31 22:59:56
595	Gold continues to shine, jumps to ₹84,900 per 10 grams;	https://news.google.com/read/CBMi1wFBVV95cUxNekpLekV6ZzFnN	2025-01-31 18:59:56
596	High volume shares on Budget day: These midcap stocks	https://news.google.com/read/CBMi_gFBVV95cUxQbEiISDVeaTVFZ	2025-02-01 10:59:56
597	Stocks to Watch on February 1: ONGC, Sun Pharma, LIC H	https://news.google.com/read/CBMi4AFBVV95cUxQpCpCHZ4bnBo	2025-01-31 23:59:56
598	What has DeepSeek done that wasn't possible for OpenAI	https://news.google.com/read/CBMi8wFBVV95cUxNwUYyYUluZVl3;	2025-01-31 19:59:56
599	ONGC Q3 Results: Net profit falls 17% to ₹8,240 crore, div	https://news.google.com/read/CBMi2AFBVV95cUxQrJInbF8tbE5VW	2025-01-31 22:59:56
600	"Kept Saying Champions Trophy 2025 Stadiums Won't Be	https://news.google.com/read/CBMi5gFBVV95cUxOdXVOZG5Wa1Jh	2025-01-31 18:59:56
601	Attack on Saif Ali Khan: Forensic report confirms identity	https://news.google.com/read/CBMi4wFBVV95cUxQcnFiUFZXF0FnH	2025-01-31 14:59:56
602	IndusInd Bank Q3 Results: Standalone PAT declines 39% Y	https://news.google.com/read/CBMi5wFBVV95cUxQNJFnUGpmMS1z	2025-01-31 15:59:56
603	IndiaAI GPU procurement sees record discounts as firms s	https://news.google.com/read/CBMi2AFBVV95cUxPVS1PaEVrWDJU	2025-01-31 12:59:56
604	Bombay High Court appoints ex-Supreme Court judge to r	https://news.google.com/read/CBMiowFBVV95cUxQU11GX05hStgyY	2025-01-31 12:59:56
605	Budget 2025 Speech LIVE: Nirmala Sitharaman proposes	https://news.google.com/read/CBMigwJBVV95cUxOaIZNc1kNkxML	2025-02-01 11:30:56
606	Vedanta Q3 Results 2025 Highlights: Net profit jumps 70%	https://news.google.com/read/CBMiIgJBVV95cUxOQJyJZGowU3hMK	2025-01-31 15:59:56
607	'In poor taste': Rashtrapati Bhavan reacts to Sonia Gandh	https://news.google.com/read/CBMipwFBVV95cUxQc3Rndk1YbIE3N	2025-01-31 18:06:21
608	Sky Force Day 8 India Box Office: Akshay Kumar, Veer Pah	https://news.google.com/read/CBMiIgJBVV95cUxNtnNyQzVvVfPck	2025-01-31 23:59:56
609	Pushpa 2 OTT release: This aspect is being talked about	https://news.google.com/read/CBMiogFBVV95cUxOcDY2UTQOMkNi	2025-01-31 17:59:56
610	Mamta Kulkarni, Laxmi Narayan Tripathi Expelled From Kin	https://news.google.com/read/CBMirwFBVV95cUxObkpDanhUV2E2	2025-01-31 12:59:56
611	Man Leaves India Despite Passport Being In Court Custod	https://news.google.com/read/CBMiLwFBVV95cUxQODRMM3VMSH	2025-02-01 11:59:56
612	Nifty 50 Prediction For Tomorrow, Feb 1 (Budget Day): Bre	https://news.google.com/read/CBMiHwJBVV95cUxNVI9YTDhNYjNIQ	2025-01-31 17:59:56
613	Huge asteroid moving towards Earth, may strike India in...	https://news.google.com/read/CBMi5AFBVV95cUxNsmk1X0tYtmFk	2025-01-30 11:59:56
614	Union Budget 2025 Live Updates: Nirmala Sitharaman Pre	https://news.google.com/read/CBMi0gFBVV95cUxPYmg5UETPSjNpT	2025-02-01 11:04:59
615	Gold prices to decline, silver to rise in 2025: Economic Sur	https://news.google.com/read/CBMiWAFBVV95cUxOTnlkUEVY1A0dI	2025-01-31 16:59:59
616	List of dead soon, verifying reports of second incident: Ku	https://news.google.com/read/CBMi3AFBVV95cUxOU1BKmi02SVdQ	2025-02-01 07:59:59
617	PM Modi hints at special provisions for poor, middle classe	https://news.google.com/read/CBMiYAFBVV95cUxQQTNwXZVvDeo2	2025-02-01 03:59:59
618	In a setback to AAP, 7 MLAs quit party ahead of Delhi polls	https://news.google.com/read/CBMioAFBVV95cUxNQI9ndmhkejVaM	2025-01-31 18:59:59
619	Delhi fog today: Visibility near zero in NCR; airport issues	https://news.google.com/read/CBMi5gFBVV95cUxONzUteGNqSWRl	2025-02-01 07:59:59
620	Ghaziabad shops, cars damaged in blasts as truck carryin	https://news.google.com/read/CBMiZAFBVV95cUxOUneEwIGWkt4C	2025-02-01 06:59:59
621	Budget 2025: FM Sitharaman announces measures aimed	https://news.google.com/read/CBMi-ABVV95cUxQV0JWSlowbkZ2I	2025-02-01 11:14:59

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 `PostgreSQL` 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



```

1 import psycopg2 as psycopg
2 import logging
3 from difflib import SequenceMatcher
4
5 class DuplicationChecker:
6     __slots__ = '_connection', '_db_configuration', '_cursor', '_log'
7     def __init__(self, config_file: dict) -> None:
8         self._db_configuration = config_file['database']
9         self._connection = psycopg.connect(
10             dbname = self._db_configuration['dbname'],
11             user = self._db_configuration['user'],
12             password = self._db_configuration['password'],
13             host = self._db_configuration['host'],
14             port = self._db_configuration['port']
15         )
16         self._cursor = self._connection.cursor()
17         self._log = logging.getLogger('DuplicateChecker')
18
19     def check_similarity(self, headline1: list, headline2: list) -> float:
20         return SequenceMatcher(None, headline1.lower(), headline2.lower()).ratio()
21
22     def is_duplicate(self, story: list, same_day_threshold: float = 0.90, different_day_threshold:
23         float = 0.95):
24         try:
25             story_date = story['Date'][:10] # Extract YYYY-MM-DD
26
27             # First check same day articles with lower threshold
28             self._cursor.execute("""
29                 SELECT publication_date, headline
30                 FROM news_articles
31                 WHERE SUBSTRING(publication_date, 1, 10) = %s
32             """,
33                 (story_date,))
34
35             same_day_stories = self._cursor.fetchall()
36
37             for existing_headline, _ in same_day_stories:
38                 similarity = self.check_similarity(story['Headline'], existing_headline)
39                 if(similarity >= same_day_threshold):
40                     self._log.info(f"Duplicate found - Headline: {story['Headline']} - Same day
41 duplicate found - Similarity: {similarity:.2f}")
42                     return True, f"Same day duplicate found - Similarity: {similarity:.2f}"
43
44             # Then check other days with higher threshold
45             self._cursor.execute("""
46                 SELECT headline, publication_date
47                 FROM news_articles
48                 WHERE SUBSTRING(publication_date, 1, 10) != %s
49             """,
50                 (story_date,))
51
52             different_day_stories = self._cursor.fetchall()
53
54             for existing_headline, _ in different_day_stories:
55                 similarity = self.check_similarity(story['Headline'], existing_headline)
56                 if( similarity >= different_day_threshold):
57                     self._log.info(f"Duplicate found - Headline: {story['Headline']} - Different day
58 duplicate found - Similarity: {similarity:.2f}")
59                     return True, f"Different day duplicate found - Similarity: {similarity:.2f}"
60
61             return False, "No duplicate found"
62
63         except Exception as error:
64             self._log.error(f"Error checking duplication: {error}")
65             return False, f"Error: {str(error)}"
66
67     def close(self):
68         self._cursor.close()
69         self._connection.close()
70
71 # ----- Code to test Module 5 -----
72
73 import yaml
74 if __name__ == "__main__":
75     with open('config.yaml', 'r') as file:
76         config = yaml.safe_load(file)
77
78     checker = DuplicationChecker(config)

```

```

79
80     test_story = {
81         "Headline": "Silicon Valley consortium values London Spirit at 295 million in Hundred coup",
82         "Thumbnail_url": "https://news.google.com/api/attachments/
CC8iLONnNUpVemhzVWpoRVRTMXZUVjllLVFJDZkF4ampCU2dLTWdrQlVJeEJ2YVV0a2dF=-w280-h168-p-df",
83         "Article_url": "https://news.google.com/read/
CBMivAFBVV95cUxNSWV1TWdWNThDZnYxbE8wdDA5NXZozZHdIX1FoT3J2SXF1c2R6ZUdFRjRlUTlGcUdFMmVBTTE4Yk5LMj10RUhYUkExWX
?hl=en-IN&gl=IN&ceid=IN%3Ae",
84         "Date": "2025-02-01 01:52:08"
85     }
86
87     is_duplicate, reason = checker.is_duplicate(test_story)
88     print(f"Reason: {reason}")
89
90     checker.close()

```

## Environment

This script uses *psycopyg* 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

```

1  import logging
2  from datetime import datetime
3  import traceback
4
5  from .module1 import *
6  from .module2 import *
7  from .module3 import *
8  from .module4 import *
9
10 def setup_logging() -> None:
11     logging.basicConfig(
12         filename = 'pipeline.log',
13         level = logging.INFO,
14         format = '%(asctime)s - %(levelname)s - %(message)s',
15         datefmt = '%Y-%m-%d %H:%M:%S'
16     )
17     console_handler = logging.StreamHandler()
18     console_handler.setLevel(logging.INFO)
19     logging.getLogger().addHandler(console_handler)
20
21 def run_pipeline(config):
22     try:
23         start_time = datetime.now()
24         logging.info("\nStarting news scrapping pipeline...\n")
25
26         # Module 1: Scrape homepage
27         home_scraper = WebPageScraper('config.yaml')
28         homepage_content = home_scraper.scrape()
29         if(not homepage_content):
30             raise Exception("\nFailed to scrape webpage\n")
31
32         # Module 2: Extract top stories link
33         top_stories_scraper = ContentScraper(homepage_content, config)

```

```

34     top_stories_url = top_stories_scraper.find_sub_heading()
35     if not top_stories_url:
36         raise Exception("\nFailed to find top stories url\n")
37
38     # Module 3: Extract stories
39     story_extractor = ContentExtractor(config)
40     stories = story_extractor.extract_stories(top_stories_url)
41     if not stories:
42         raise Exception("\nNo stories were extracted\n")
43
44     # Module 4: Store in database
45     db = NewsDB(config)
46     stored_count = 0
47     skipped_count = 0
48
49     for story in stories:
50         if db.store_article(story):
51             stored_count += 1
52         else:
53             skipped_count += 1
54
55     db.close()
56
57     end_time = datetime.now()
58     duration = (end_time - start_time).total_seconds()
59     logging.info(f"\nPipeline completed. Stored: {stored_count}, Skipped: {skipped_count},
Duration: {duration:.2f}s\n")
60     return True
61
62 except Exception as error:
63     logging.error(f"\nPipeline failed:\t{str(error)}\n")
64     logging.error(traceback.format_exc())
65     return False
66
67 # ----- Code to test Module 6 -----
68
69 import yaml
70 import time
71
72 if __name__ == "__main__":
73     setup_logging()
74     try:
75         with open('config.yaml', 'r') as file:
76             config = yaml.safe_load(file)
77
78         frequency = config.get('pipeline', {}).get('frequency', ) # Default: .5 hour in seconds
79
80         while True:
81             run_pipeline(config)
82             logging.info(f"Waiting {frequency} seconds until next run...")
83             time.sleep(frequency)
84
85     except KeyboardInterrupt:
86         logging.info("Pipeline stopped by user")
87     except Exception as error:
88         logging.error(f"Fatal error: {str(error)}")
89         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.

```

1 .
2     DA24M011_Assignment1
3         __init__.py
4         __main__.py
5         __pycache__
6         config.yaml

```

```

7         duplicates.log
8         module1.py
9         module2.py
10        module3.py
11        module4.py
12        module5.py
13        module6.py
14        pipeline.log
15        stories.json
16    DA5402_MLOPS_A1.pdf
17    create_table.sql
18    directory_structure.txt
19    mlops_a1
20        bin
21        include
22        lib
23        pyvenv.cfg
24    pipeline.log
25    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.

```

1 # Base configuration for Google News Scraper
2 webpage:
3     base_url: "https://news.google.com"
4     user_agent: "Mozilla/5.0 AppleWebKit/537.36"
5     sub_heading: "Top stories"
6
7 database:
8     dbname: "nandhakishorecs"
9     user: "nandhakishorecs" #replace with your computer's username
10    password: "" #replace with your computer's password
11    host: "localhost"
12    port: 5432
13
14 pipeline:
15     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.