```
## packages required : selenium
# !pip install selenium
```

This notebook is used for debugging and initial creation of code

Mainly created to play around with and all code in this NB has been converted to separate python modules

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.options import Options

from bs4 import BeautifulSoup
import time
from tqdm.notebook import tqdm
import hashlib

import re
from PIL import Image
from bson.binary import Binary

from pymongo import MongoClient
from bson.binary import Binary
import os
```

Module 1

```
# getting the webdriver
driver = webdriver.Chrome()  # Make sure you have ChromeDriver
installed

chrome_options = Options()
# chrome_options.add_argument("--headless=new")
# chrome_options.add_argument("--force-device-scale-factor=1")  #
Prevent DPI scaling issues
# chrome_options.add_argument("--window-size=1920,1080")  # Mandatory
for consistent rendering

## loading the page
url = "https://news.google.com/"
driver.get(url)

# simulating scrolling for webpage and load-more button

# To check whether the loading has finished or not
def is_page_loaded(driver):
    return driver.execute_script("return document.readyState") ==
"complete"
```

```
# getting the last scroll height
last_height = driver.execute_script("return
document.body.scrollHeight")

# maximum scroll iterations allowed is 10
scroll_iterations = 0

while scroll_iterations < 5:

    driver.execute_script("window.scrollTo(0,
document.body.scrollHeight);")

    # Loop to check loading status and we will wait for maximum of 3
seconds to load the page
    timeout = 3  # seconds
    start_time = time.time()

    while not is_page_loaded(driver):
        if time.time() - start_time > timeout:
            print("Timeout reached. The page is still loading.")
            break

    # checking for load more button as well
    try:
        load_more_button = driver.find_element(By.CLASS_NAME, "load-
more-button")
        load_more_button.click()
        time.sleep(2)
    except : pass

    # getting the new height
    new_height = driver.execute_script("return
document.body.scrollHeight")

    # if the heights are same and the page is loaded , then we do
timeout
    if (new_height == last_height) and is_page_loaded(driver) :
        break

    # updating the height and scroll_iterations
    last_height = new_height
    scroll_iterations += 1

## Now extracting the contents using beautifulsoup
page_source = driver.page_source
soup = BeautifulSoup(page_source, 'html.parser')

## putting everything inside a fn

def module_1_web_scrapping_with_lazy_loading(driver = driver, url =
"https://news.google.com/"):
```

```
    # getting the url
    driver.get(url)
    time.sleep(1)
    # time.sleep(2)

    # simulating scrolling for webpage and load-more button
    # getting the last scroll height
    old_items = len(driver.find_elements(By.CSS_SELECTOR, ".item-
class, div, article, li"))
    last_height = driver.execute_script(
        "return Math.max(document.documentElement.scrollHeight,
document.body.scrollHeight, document.documentElement.clientHeight);")

    # maximum scroll iterations allowed is 10
    scroll_iterations = 0

    while scroll_iterations < 5:

        driver.execute_script("""window.scrollBy(0,
Math.max(document.documentElement.scrollHeight,
                            document.body.scrollHeight,
document.documentElement.clientHeight));""")
        time.sleep(1)

        # Loop to check loading status and we will wait for maximum of
5 seconds to load the page
        timeout = 5  # seconds
        start_time = time.time()

        while not is_page_loaded(driver):
            if time.time() - start_time > timeout:
                print("Timeout reached. The page is still loading.")
                break
            time.sleep(0.5)

        # checking for load more button as well
        try:
            load_more_button = driver.find_element(By.CLASS_NAME,
"load-more-button")
            load_more_button.click()
            time.sleep(2)
        except : pass

        # getting the new height and new items
        new_items = len(driver.find_elements(By.CSS_SELECTOR, ".item-
class, div, article, li"))
        new_height = driver.execute_script(
        "return Math.max(document.documentElement.scrollHeight,
document.body.scrollHeight, document.documentElement.clientHeight);")
```

```
        # print(last_height, new_height, old_items, new_items)

        # if the heights are same and the page is loaded , then we do
timeout
        if (new_height == last_height) and is_page_loaded(driver) and
(new_items == old_items):
            break

        # updating the height and scroll_iterations
        last_height = new_height
        old_items = new_items
        scroll_iterations += 1

    ## Now extracting the contents using beautifulsoup
    page_source = driver.page_source
    soup = BeautifulSoup(page_source, 'html.parser')
    return soup

page_content_module_1 = module_1_web_scrapping_with_lazy_loading()
```

Module 2

```
# getting the top stories
""" Doubt to be discussed with achyutha (whether it is just stories or
top stories)"""
story_links = soup.find_all('a', href=re.compile(r'^./stories/'))

# for saving the links
top_story_links = []

# getting all the links
for links in story_links:
    story_url = f"https://news.google.com{links['href'].lstrip('.')}"
    # print(story_url)
    top_story_links.append(story_url)


def module_2_stories_link(soup, section = 'stories'):

    story_links = soup.find_all('a',
href=re.compile(f'^./{section}/'))

    # for saving the links
    top_story_links = []

    # getting all the links
    for links in story_links:
        story_url =
f"https://news.google.com{links['href'].lstrip('.')}"
```

```
        # print(story_url)
        top_story_links.append(story_url)

    print(f"Total number of stories link extracted :
{len(top_story_links)}")

    return top_story_links

m2_top_story_links = module_2_stories_link(soup =
page_content_module_1, section = 'stories')
```

```
Total number of stories link extracted : 29
```

Module 3

```
# to get the thumbnail and links from the page
m3_soup1 = module_1_web_scrapping_with_lazy_loading(driver = driver,
                                                    url =
m2_top_story_links[1])

"""
The structure of the articles in the page should be:

article (class : "MQsxIb xTewfe tXImLc R7GTQ keNKEd keNKEd VkAdve
GU7x0c JMJvke q4atFc")
    |
    |
    ------- h4 (class : "ipQwMb ekueJc RD0gLb")
    |
    |
    ------- img (class : "tvs3Id QwxBBf")


From this we will extract the image and thumbnail

##########################################################################
########################

Data extraction Loop :

-> Get the top stories link

-> For each link:
    -> extract the page
    -> for each article in the page:
        -> extract image and thumbnail

##########################################################################
########################
"""
```

```
all_news_cards = m3_soup1.find_all('article', {'class': 'MQsxIb xTewfe
tXImLc R7GTQ keNKEd keNKEd VkAdve GU7x0c JMJvke q4atFc'})
news_stories = []

for news_articles in all_news_cards:

        # Extract headline
        headline = news_articles.find('h4', {'class': 'ipQwMb ekueJc
RD0gLb'})
        headline_text = headline.get_text(strip=True) if headline else
"None"

        # Extracting thumbnails (using src or data-src attr)
        img = news_articles.find('img', {'class': 'tvs3Id QwxBBf'})
        # print(img)

        thumbnail_url = None
        if img:
            thumbnail_url = img.get('src') or img.get('data-src')
            if thumbnail_url or thumbnail_url.startswith('//'):
                thumbnail_url =
f'https://news.google.com{thumbnail_url}'

        news_stories.append({
            'headline': headline_text,
            'thumbnail': thumbnail_url
        })

news_stories[0] # sample
```

```
{'headline': 'Blackhawk pilot killed in DC plane crash identified
while trans soldier receives flak on social media: Wh',
 'thumbnail':
'https://news.google.com/api/attachments/CC8iI0NnNDRTbVpWVEZGdWFpMW5ib
GRWVFJDZkF4ampCU2dLTWdB=-w100-h100-p-df-rw'}
```

```
## Full loop code inside a fn

def module_3_thumbnail_img_extraction(driver = driver,
                                      top_stories_url =
m2_top_story_links[:3],
                                      article_class_name = "MQsxIb
xTewfe tXImLc R7GTQ keNKEd keNKEd VkAdve GU7x0c JMJvke q4atFc",
                                      headline_class_name = "ipQwMb
ekueJc RD0gLb",
                                      img_class_name = "tvs3Id
QwxBBf"):

    thumbnail_img_list = []
```

```
    print(f"Total pages available to scrap : {len(top_stories_url)}")

    for url in tqdm(top_stories_url):

        # extracting the contents of all pages
        top_story_page_content =
module_1_web_scrapping_with_lazy_loading(driver = driver,
                                                        url = url)

        all_news_cards = top_story_page_content.find_all('article',
{'class': article_class_name})

        for news_articles in all_news_cards:

                # Extract headline
                headline = news_articles.find('h4', {'class':
headline_class_name})
                headline_text = headline.get_text(strip=True) if
headline else "None"

                # Extracting thumbnails (using src or data-src attr)
                img = news_articles.find('img', {'class':
img_class_name})
                # print(img)

                thumbnail_url = None
                if img:
                    thumbnail_url = img.get('src') or img.get('data-
src')
                    if thumbnail_url or
thumbnail_url.startswith('//'):
                        thumbnail_url =
f'https://news.google.com{thumbnail_url}'

                thumbnail_img_list.append({
                    'headline': headline_text,
                    'thumbnail': thumbnail_url
                })

    return thumbnail_img_list

m3_thumb_img_test = module_3_thumbnail_img_extraction()
```

```
Total pages available to scrap : 3
```


```
{"model_id":"e794d0ff71934aa8bea2277c614c950a","version_major":2,"vers
ion_minor":0}
```


```
print(f"total datapoints extracted : {len(m3_thumb_img_test)}")
m3_thumb_img_test[0] # example
```

```
total datapoints extracted : 166

{'headline': 'India’s budget gives tax relief to middle class to boost spending, growth',
 'thumbnail': 'https://news.google.com/api/attachments/CC8iK0NnNTNkMVJpTldoV1ZGbDBNWFZpVFJEZ0F4aUFCU2dLTWdhWlFKZ1NwUWM=-w100-h100-p-df-rw'}

## downloading and saving all images
"""
root_img - root path (folder) where all images are stored

"""
import requests
root_img = "images"

def download_and_store_image(heading_url_data, root_img = "images"):

    thumbnail_heading_dataset = []
    base_val = int(os.listdir(root_img)[-1].split('.')[0].split('_')[-1]) if len(os.listdir(root_img)) > 0 else 0

    for i,data_point in tqdm(enumerate(heading_url_data)):

        # thumbnail and headline
        image_url = data_point['thumbnail']
        image_headline = data_point['headline']
        image_id = (base_val + i + 1) # (this will be the name of the image as well)

        # getting the image and storing them
        img_data = requests.get(image_url).content
        with open(f"{root_img}/image_{image_id}.jpg", 'wb') as handler:
            handler.write(img_data)

        # creating a dict to store the data
        dp = {"headline" : image_headline, "image_id" : image_id, "image_url" : image_url}

        # adding things to the dataset
        thumbnail_heading_dataset.append(dp)

    return thumbnail_heading_dataset

downloaded_headline_images_data = download_and_store_image(heading_url_data = m3_thumb_img_test)

{"model_id":"c25d6efc2b1e416d8e98817b7dade0d8","version_major":2,"version_minor":0}
```

Module 4 and Module 5

My Approach for Module 4 and Module 5

-> for this we need to **hash** the image using `hashlib` package in python

-> *Table 1 (headline table) :* contains

- headline,
- image url,
- image id,
- image_index (row index of the image in image Table),
- image hash
- (other metadata as well , if required)

-> *Table 2 (image table) :*

- Has the hash of the image + hash of the headline
- bin of the image as well , if required (here I am storing the bin as well as it these are small images)

-> *Logic :*

- We can get the image from the ***image_id***, which is the name of the image and stored some other folder (from this we can get the image)
- **Hash is there to ensure whether the images and headlines are same** , we will get the image and check its hash with the corresponding row in the images table
- while storing images and headlines, if we get same has then we wont store it

```
# Connect to MongoDB
client = MongoClient('mongodb://localhost:27017/')

# Create/access database and collection (table)
db = client['image_text_db']

# creating img_table and headline_table
img_table = db['thumbnail_table']
headline_table = db['headline_table']

# putting the hash table as unique, so we wont allow duplicates as
well
img_table.create_index([('image_headline_hash', 1)], unique=True)
```

```
'image_hash_1'
```

```
# iterating and storing all the images_hash and headlines

def module_4_and_5_store_in_database(root_img,
                                     extracted_data,
                                     headline_table,
                                     img_table):
```

```
    for i,datapoint in tqdm(enumerate(extracted_data)):

        current_headline = datapoint['headline']
        current_image_id = datapoint['image_id']
        current_url = datapoint['image_url']
        current_headline_hash =
hashlib.sha256(current_headline.encode('utf-8')).hexdigest()

        # getting the image id
        img_path = f"{root_img}/image_{current_image_id}.jpg"

        # Reading the image data
        with open(img_path, 'rb') as current_img_data:
            # creating the hash of the image
            current_img_hash =
hashlib.sha256(current_img_data.read()).hexdigest()
            binary_image = Binary(current_img_data.read())

        # Create table row document
        headline_document = {
            "headline": current_headline,
            "image_url": current_url,
            "image_id": current_image_id,
            "image_index": img_table.count_documents({}),
        }

        img_document = {"image_headline_hash" : current_img_hash +
current_headline_hash,
                        "image_bin" : binary_image}

        try:
            # Insert into table of headline and images
            img_table.insert_one(img_document)
            headline_table.insert_one(headline_document)

        except: pass

    print(f"Successfully stored {img_table.count_documents({})}
records")

module_4_and_5_store_in_database(root_img = root_img,
                                 extracted_data =
downloaded_headline_images_data,
                                 headline_table = headline_table,
                                 img_table = img_table)
```

{"model_id":"407299dd1ab042f2b596ab8cc4750648","version_major":2,"version_minor":0}

```
Successfully stored 117 records
```

Module 6

```
import m1
import m2
import m3
import m4_5

def module_6_orchestrator(url = "https://news.google.com/", # for m1
                          section = 'stories', # for m2
                          article_class_name = "MQsxIb xTewfe tXImLc
R7GTQ keNKEd keNKEd VkAdve GU7x0c JMJvke q4atFc", # for m3
                          headline_class_name = "ipQwMb ekueJc
RD0gLb", # for m3
                          img_class_name = "tvs3Id QwxBBf", # for m3
                          host = "mongodb://localhost:27017/", # for
m4
                          root_img = "images", # for m4 and m3
                          ):

    # getting the base google news page
    google_news_page_soup =
m1.module_1_web_scrapping_with_lazy_loading(url = url)

    # extracting the links from the soup
    top_story_links = m2.module_2_stories_link(soup =
google_news_page_soup, section = section)[:2]

    # now extracting stories , thumbnail from that
    headline_thumbnail_url_list =
m3.module_3_thumbnail_img_extraction(top_stories_url =
top_story_links,

article_class_name = article_class_name,

headline_class_name = headline_class_name,

img_class_name = img_class_name)

    print(f"{len(headline_thumbnail_url_list)} headline and thumbnail
urls have been collected")

    # now downloading and saving those images
    headline_downloaded_img =
m3.download_and_store_image(heading_url_data =
headline_thumbnail_url_list,
                                                        root_img =
root_img)

    print(f"{len(headline_downloaded_img)} has been downloaded
successfully")
```

```
    # now storing them to the database
    headline_table, img_table = m4_5.connect_database(host = host)
    prev_table_size = headline_table.count_documents({})

    # storing in database
    m4_5.module_4_and_5_store_in_database(root_img = root_img,
                                      extracted_data =
headline_downloaded_img,
                                      headline_table = headline_table,
                                      img_table = img_table)

    curr_table_size = headline_table.count_documents({})

    print(f"{curr_table_size - prev_table_size} rows has been
populated in database")

# module_6_orchestrator()

url = "https://news.google.com/" # for m1
section = 'stories' # for m2
article_class_name = "MQsxIb xTewfe tXImLc R7GTQ keNKEd keNKEd VkAdve
GU7x0c JMJvke q4atFc" # for m3
headline_class_name = "ipQwMb ekueJc RD0gLb" # for m3
img_class_name = "tvs3Id QwxBBf" # for m3
host = "mongodb://localhost:27017/" # for m4
root_img = "images" # for m4 and m3

# getting the base google news page
google_news_page_soup =
m1.module_1_web_scrapping_with_lazy_loading(url = url)

# extracting the links from the soup
top_story_links = m2.module_2_stories_link(soup =
google_news_page_soup, section = section)[:1]

# now extracting stories , thumbnail from that
headline_thumbnail_url_list =
m3.module_3_thumbnail_img_extraction(top_stories_url =
top_story_links,

article_class_name = article_class_name,

headline_class_name = headline_class_name,

img_class_name = img_class_name)

print(f"{len(headline_thumbnail_url_list)} headline and thumbnail urls
have been collected")

# now downloading and saving those images
```

```
headline_downloaded_img = m3.download_and_store_image(heading_url_data
= headline_thumbnail_url_list,
                                                      root_img =
root_img)

print(f"{len(headline_downloaded_img)} has been downloaded
successfully")

# now storing them to the database
headline_table, img_table = m4_5.connect_database(host = host)
prev_table_size = headline_table.count_documents({})

# storing in database
m4_5.module_4_and_5_store_in_database(root_img = root_img,
                                      extracted_data =
headline_downloaded_img,
                                      headline_table = headline_table,
                                      img_table = img_table)

curr_table_size = headline_table.count_documents({})

print(f"{curr_table_size - prev_table_size} rows has been populated in
database")
```

```
1499 4682 432 1229
4682 4682 1229 1321
4682 4682 1321 1321
Total number of stories link extracted : 26
Total pages available to scrap : 1

  0%|          | 0/1 [00:00<?, ?it/s]

4646 4646 1239 1239

100%|██████████| 1/1 [00:03<00:00,  3.43s/it]

53 headline and thumbnail urls have been collected

53it [00:06,  8.22it/s]

53 has been downloaded successfully

53it [00:00, 105.73it/s]

Successfully stored 53 records
53 rows has been populated in database
```

```
import logging

# Create and configure logger
```

```
logging.basicConfig(filename="logs_file.log",
                    format='%(asctime)s %(message)s',
                    filemode='w')

# Creating an object
logger = logging.getLogger()

# Setting the threshold of logger to DEBUG
logger.setLevel(logging.DEBUG)

# Test messages
p = 90
# logger.debug(f"Harmless debug Message")
logger.info(f"Just an information {p}")
# logger.warning("Its a Warning")
```

creating a YAML file for configuration

```
import yaml
import io

# Define data
params = {
"url" : "https://news.google.com/", # for m1
"section" : 'stories', # for m2
"article_class_name" : "MQsxIb xTewfe tXImLc R7GTQ keNKEd keNKEd
VkAdve GU7x0c JMJvke q4atFc", # for m3
"headline_class_name" : "ipQwMb ekueJc RD0gLb", # for m3
"img_class_name" : "tvs3Id QwxBBf", # for m3
"host" : "mongodb://localhost:27017/", # for m4
"root_img" : "images", # for m4 and m3
}

# Write YAML file
with io.open('All_Parameters.yaml', 'w', encoding='utf8') as outfile:
    yaml.dump(params, outfile, default_flow_style=False,
allow_unicode=True)

# Read YAML file
with open('All_Parameters.yaml', 'r') as stream:
    data_loaded = yaml.safe_load(stream)

True
```