**Aim**

Select any social media platform of your choice and perform scraping, crawling, and parsing.

**Software Requirements**

* **Operating System:** Windows, macOS, or Linux
* **Programming Language:** Python 3.x
* **Libraries Required:**
  + **requests** - For sending HTTP requests to fetch webpage content
  + **BeautifulSoup** - For parsing and extracting HTML elements
  + **Feedparser** - For extracting RSS feed data from Medium
  + **Pandas** - For storing and manipulating extracted data
* **Development Environment:** Jupyter Notebook, VS Code, or PyCharm

**Theory**

**Introduction to Web Scraping, Crawling, and Parsing**

* **Web Scraping:** The process of extracting structured data from websites.
* **Web Crawling:** Systematic browsing of a website to discover and collect links and data.
* **Parsing:** Transforming raw HTML content into meaningful and structured information.

Medium is a content-rich blogging platform where articles are categorized by tags. Extracting Medium articles enables insights into trending topics, author popularity, and content analysis. Since Medium has limited API access, web scraping is a viable alternative.

**Tools & Technologies Used**

* **Python** – Programming language for data extraction.
* **Requests** – To send HTTP requests and fetch webpage content.
* **BeautifulSoup** – For parsing and extracting HTML elements.
* **Feedparser** – For extracting RSS feed data from Medium.
* **Pandas** – To store and manipulate extracted data.

**Steps to Perform Web Scraping on Medium**

**Step 1: Identify the Target Data**

* Choose a specific topic (e.g., Python articles).
* Identify elements to scrape: Titles, Authors, URLs, and Publication Dates.

**Step 2: Check Medium's robots.txt**

Before scraping, check [Medium's robots.txt](https://medium.com/robots.txt) to ensure compliance with scraping rules.

**Step 3: Inspect the Web Page Structure**

Use browser DevTools (Ctrl+Shift+I in Chrome) to analyze HTML tags and locate relevant elements.

**Step 4: Implement the Web Scraper**

The following Python script extracts articles from Medium’s RSS feed, retrieves full content, and filters the extracted data.

**Code**

import feedparser

import pandas as pd

import requests

from bs4 import BeautifulSoup

import time

import re

import os

from langdetect import detect, DetectorFactory

DetectorFactory.seed = 0

def clean\_html(raw\_html):

"""Remove HTML tags and return plain text."""

return BeautifulSoup(raw\_html, "html.parser").get\_text()

def is\_english(text):

"""Check if the given text is in English."""

try:

return detect(text) == "en"

except:

return False

def get\_medium\_full\_article(url):

"""Fetch full article content while filtering out unwanted text."""

headers = {

"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36"

}

try:

response = requests.get(url, headers=headers, timeout=10)

if response.status\_code != 200:

return f"Failed to fetch article: {response.status\_code}"

soup = BeautifulSoup(response.text, "html.parser")

article\_body = soup.find("article")

if not article\_body:

return "Article content not found (may be paywalled)."

paragraphs = article\_body.find\_all("p")

article\_text = "\n".join([p.get\_text() for p in paragraphs])

unwanted\_patterns = [r'\bFollow\b', r'\bListen\b', r'\bShare\b', r'\bSign up\b', r'\bSign in\b']

for pattern in unwanted\_patterns:

article\_text = re.sub(pattern, '', article\_text, flags=re.IGNORECASE)

return article\_text.strip()

except Exception as e:

return f"Error fetching article: {str(e)}"

def get\_filtered\_medium\_articles(rss\_url, topic, articles\_per\_topic=5):

"""Fetch Medium articles from RSS, filter by topic, and extract full content."""

feed = feedparser.parse(rss\_url)

all\_articles = []

seen\_links = set()

count = 0

for entry in feed.entries:

if count >= articles\_per\_topic:

break

full\_content = get\_medium\_full\_article(entry.link)

summary = clean\_html(entry.summary)

if is\_english(full\_content) and is\_english(summary):

article = {

"topic": topic,

"title": entry.title,

"link": entry.link,

"published": entry.published,

"summary": summary,

"full\_content": full\_content

}

all\_articles.append(article)

seen\_links.add(entry.link)

count += 1

time.sleep(2)

return all\_articles

topics = ["Technology", "Sports", "Business", "Science"]

rss\_urls = {topic: f"https://medium.com/feed/tag/{topic.lower().replace(' ', '-')}" for topic in topics}

articles\_per\_topic = 50

filtered\_articles = []

for topic, rss\_url in rss\_urls.items():

print(f"Fetching articles for: {topic}")

filtered\_articles.extend(get\_filtered\_medium\_articles(rss\_url, topic, articles\_per\_topic))

df = pd.DataFrame(filtered\_articles)

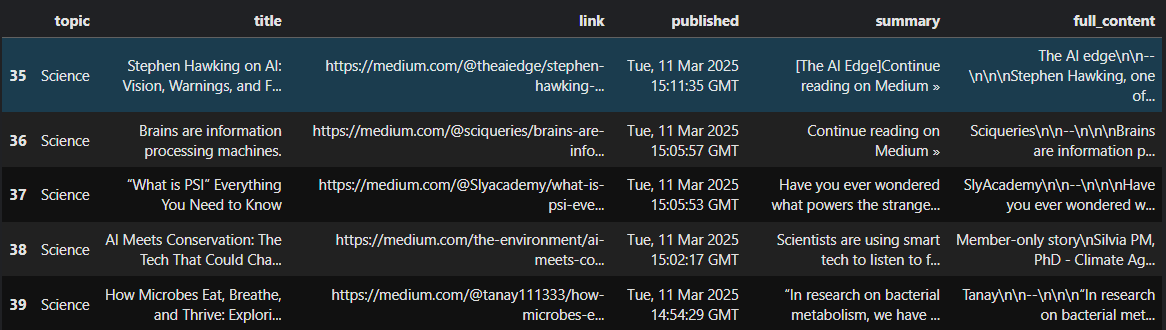
csv\_file = "medium\_articles\_filtered.csv"

df.to\_csv(csv\_file, mode='a', index=False, header=not os.path.exists(csv\_file))

print(f"Saved {len(df)} new English articles to {csv\_file}")

df.head()

**Output**

****

**Conclusion**

Through web scraping, crawling, and parsing, we successfully extracted article details from Medium’s RSS feeds and web pages. The extracted data can be used for content analysis, trend detection, and sentiment analysis while adhering to ethical web scraping guidelines.