# Implementation Intuition

1. Use requests, BeautifulSoup for an access to sciencedaily.com
2. Starts off with one article and went through the page source information for each section of the article
3. Then expand from a single article to the articles under single category of the page
4. Expand again to the entire categories on sciencedaily.com
5. When processing step 4, because of the large number of articles to send request and retrieve data, it takes longer to complete. To resolve this, ThreadPoolExecutor was added to send multiple requests concurrently.
6. For the types of information to retrieve, I assumed the dataset is for producing scientific articles in machine learning.

# Approach Idea : Take individual HTML content and/or with Meta tags

For each section of the article, check HTML headers, class, and tags to retrieve data

```
def get_article_data(article_url):
  full_article_url = f"{base_url}{article_url}"
  article_soup = get_soup(full_article_url)

  title = article_soup.find(class_= 'headline').get_text(strip=True)
  date =  article_soup.find('dd', {"id" :'date_posted'}).get_text(strip=True)
  source = article_soup.find('dd', {"id" : "source"}).get_text()
  abstract = article_soup.find('dd', {"id" : "abstract"}).get_text(strip=True)
  category = article_soup.find('ul', {"id" : "related_topics"}).find('a').get_text(strip=True)
  keywords = article_soup.find('ul', {"id" : "related_terms"}).get_text(strip=True, separator=', ')
  story_text = find_fullstory(article_soup)
  reference = find_doi(article_soup)

  return ({
      'url' : full_article_url,
      'title' : title,
      'date' : date,
      'source' : source,
      'summary' : abstract,
      'category' : category,
      'full story' : story_text,
      'reference' : reference
  })
```

For the general contents such as title, abstract, keywords and url, use meta tags

```
summary_tag = article_soup.find('meta', attrs={'name':'description'}).get('content', ' ')
final_article['summary'] = summary_tag

keywords_tag = article_soup.find('meta', attrs={'name':'keywords'}).get('content', ' ')
final_article['keywords'] = keywords_tag

url_tag = article_soup.find('meta', attrs={'property':'og:url'}).get('content', ' ')
final_article['url'] = url_tag
```

# Challenge : Parsing specific data is done with manual-manner

**Pros :**

- take necessary information only and discard the others
- If dataset has specific purpose (ex. Generating scientific article with AI), then retrieving the data in such way won't be a huge bottleneck.

**Cons :**

- Lack of completeness
- some information is missing in the dataset and to add one, manually type in the conditions.

Need to balance out between
"Add necessary data to the blank space" and "Remove unnecessary data from the complete pool"

```
title = article_soup.find(class_= 'headline').get_text(strip=True)
date =  article_soup.find('dd', {"id" :'date_posted'}).get_text(strip=True)
source = article_soup.find('dd', {"id" : "source"}).get_text()
abstract = article_soup.find('dd', {"id" : "abstract"}).get_text(strip=True)
category = article_soup.find('ul', {"id" : "related_topics"}).find('a').get_text(strip=True)
keywords = article_soup.find('ul', {"id" : "related_terms"}).get_text(strip=True, separator=', ')
story_text = find_fullstory(article_soup)
reference = find_doi(article_soup)
```

# Improvement Idea

Idea of complete data extraction :

Take the entire page as text to not to miss out any information

```
soup = BeautifulSoup(res.text, 'html.parser')

for script in soup(["script", "style"]):
  script.extract()

text = soup.get_text()
lines = (line.strip() for line in text.splitlines())
# break multi-headlines into a line each
chunks = (phrase.strip() for line in lines for phrase in line.split("  "))
# drop blank lines
text = '\n'.join(chunk for chunk in chunks if chunk)
```

Challenge :

Data are all in text format and there is no classifier to distinguish which text falls into the other.

Idea to remove unnecessary text:

- Take the entire information on the website as text format
- Set the list of keywords that are relevant to the purpose of the dataset
- Filter out the text using the keywords list

OR,

- Check tags, class, headers of the data that are explicitly irrelevant and remove them before processing

```
# Example: Remove specific divs or sections by class names
for class_name in ['advertisement', 'related-articles', 'footer', 'header']:
    for tag in soup.find_all('div', class_=class_name):
        tag.decompose()
```

# Other improvement in current code

Exception handling:

Better error handling is necessary for articles without component values - some articles contain reference and some do not. To not to discard the data, add error handling conditions for each part of the data

Remove duplicate lines of code:

Multiple requests and responses to the different urls are being done for each article of the webpage.

Resolve text decoding & encoding:

Some articles contain text other than English hence it was necessary to check Unicode sequence for such characters.

# Reference to check

For further development on this,

I tried to find research on generating datasets from academic webpages such as arxiv.org

The project below gives some idea about using such webpage as a dataset

## On the Use of ArXiv as a Dataset

**Colin B. Clement**
Cornell University, Department of Physics
Ithaca, New York 14853-2501, USA
cc2285@cornell.edu

**Matthew Bierbaum**
Cornell University, Department of Information Science
Ithaca, New York 14853-2501, USA
mkb72@cornell.edu

**Kevin O'Keeffe**
Senseable City Lab, Massachusetts Institute of Technology
Cambridge, MA 02139
kokeeffe@mit.edu

**Alexander A. Alemi**
Google Research
Mountain View, CA
alemi@google.com

https://github.com/mattbierbaum/arxiv-public-datasets/tree/master?tab=readme-ov-file