**Coventry University**

**Faculty of Engineering, Environment and Computing**

**United Kingdom**



**STW7071CEM: Information Retrieval**

Submitted by

**Dipin Baral**

Date of Submission:

25th March, 2025

**Table of Contents**

**Introduction** 4

Crawler 4

Classification 11

Data preparation 11

**Search Engine for Crawler and Classification** 16

Crawler 16

Classification 17

Output for app.py 18

**User-Interface** 19

Classify 19

Crawler 20

**Conclusion** 20

**References** 21

**Appendix** 22

**List of figures**

Figure 1: - Crawler architecture 4

Figure 2: - Crawling publication & authors 5

Figure 3: - BaseURL 5

Figure 4: - Crawling publication & authors 6

Figure 5: - Authors 7

Figure 6: - Year 7

Figure 7: - Crawling year 7

Figure 8: - Saving json file 8

Figure 9: - Schedule for crawler 8

Figure 10: - Loading json file 9

Figure 11: - Data Cleaning and Transformation for Publications Dataset 10

Figure 12: - Saving data process into csv file 10

Figure 13: - bbc.csv file 11

Figure 14: - Loading bbc.csv file 11

Figure 15: - Text processing and Data Splitting function 12

Figure 16: - Model Evaluation, Saving, and Text Classification 13

Figure 17: - Remove Nan values 13

Figure 18: - Preprocessing and Data Splitting 14

Figure 19: - Saving and Using Pretrained Models 15

Figure 20: - Use of nltk function 16

Figure 21: - Crawler in app.py 16

Figure 22: - Classification in app.py 17

Figure 23: - Route to main.html in app.py 18

Figure 24: - start app.py 18

Figure 25: - main.html for crawler and classification 19

Figure 26: - classification output 19

Figure 27: - Crawler output 20

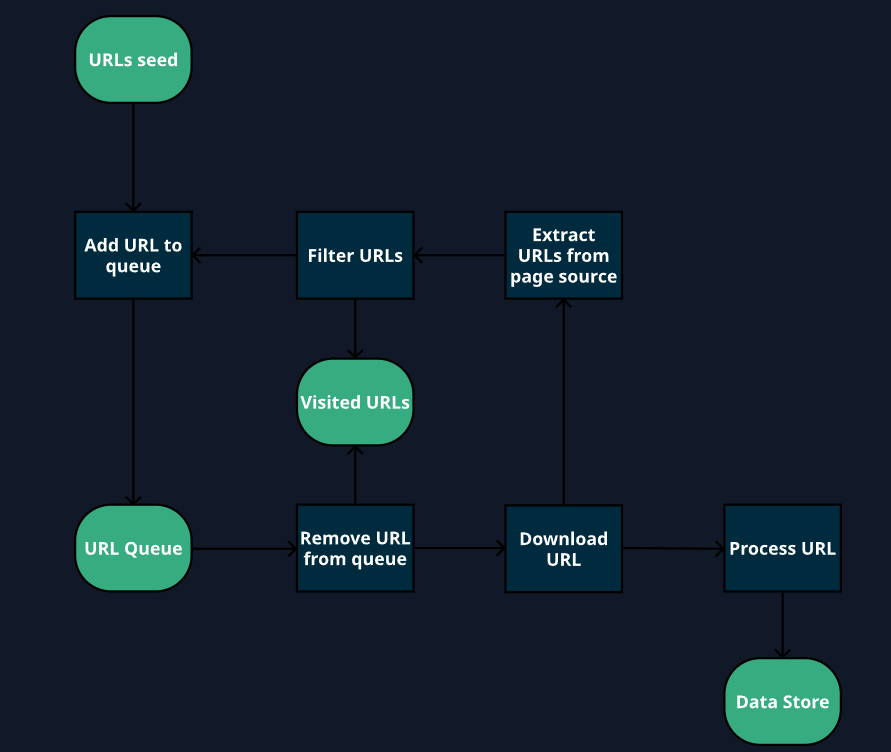
**Introduction**

This course comprises two primary assignments designed to improve knowledge of information retrieval and Subject Classification. Building a vertical search engine centered on scholarly publications from Coventry University's School of Economics, Finance, and Accounting is the aim of Task 1. Academic staff profiles on the university's "Pure Portal" will be crawled by the system, which will extract pertinent data like authors, publication titles, and years. This data will be processed and indexed by the search engine, allowing users to query and retrieve results that are restricted to works written or co-authored by department members. Using an automated crawler, the search results will be updated on a regular basis and ranked according to relevancy, much like Google Scholar. Users will be able to view the pertinent links to the publications and submit queries through a straightforward user interface.

Creating a document classification system that can group text into predetermined categories that are politics, business, and health—is the aim of Task 2. A set of at least 90 documents saved as bbc.csv from these categories will be used to train the system. The model will learn to categorize new documents. The ability of the system to handle different input types, such as short and long texts, as well as more difficult cases, will be tested in order to assess its performance. The course offers a thorough education in machine learning-based text classification as well as web scraping for information retrieval.

**Crawler**

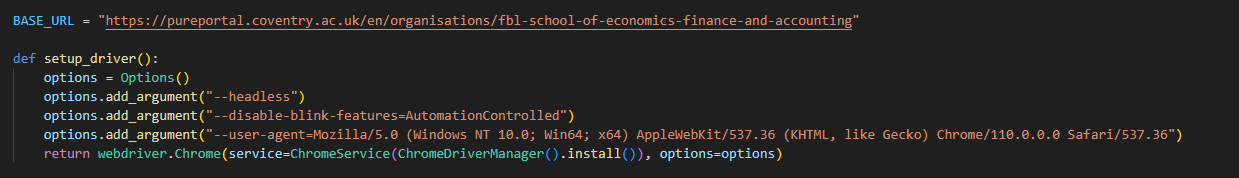
The process by which search engines employ automated bots, sometimes referred to as crawlers or spiders, to methodically search the web and index web content is called crawling. These crawlers collect information on text, images, metadata, and other elements by following links between pages. When users search for information, search engines can return pertinent results because this data is then saved in a database. For effective and precise search results, crawling makes sure search engines can maintain their indexes current with the most recent web content (Koller, 2019).



*Figure 1: - Crawler architecture*

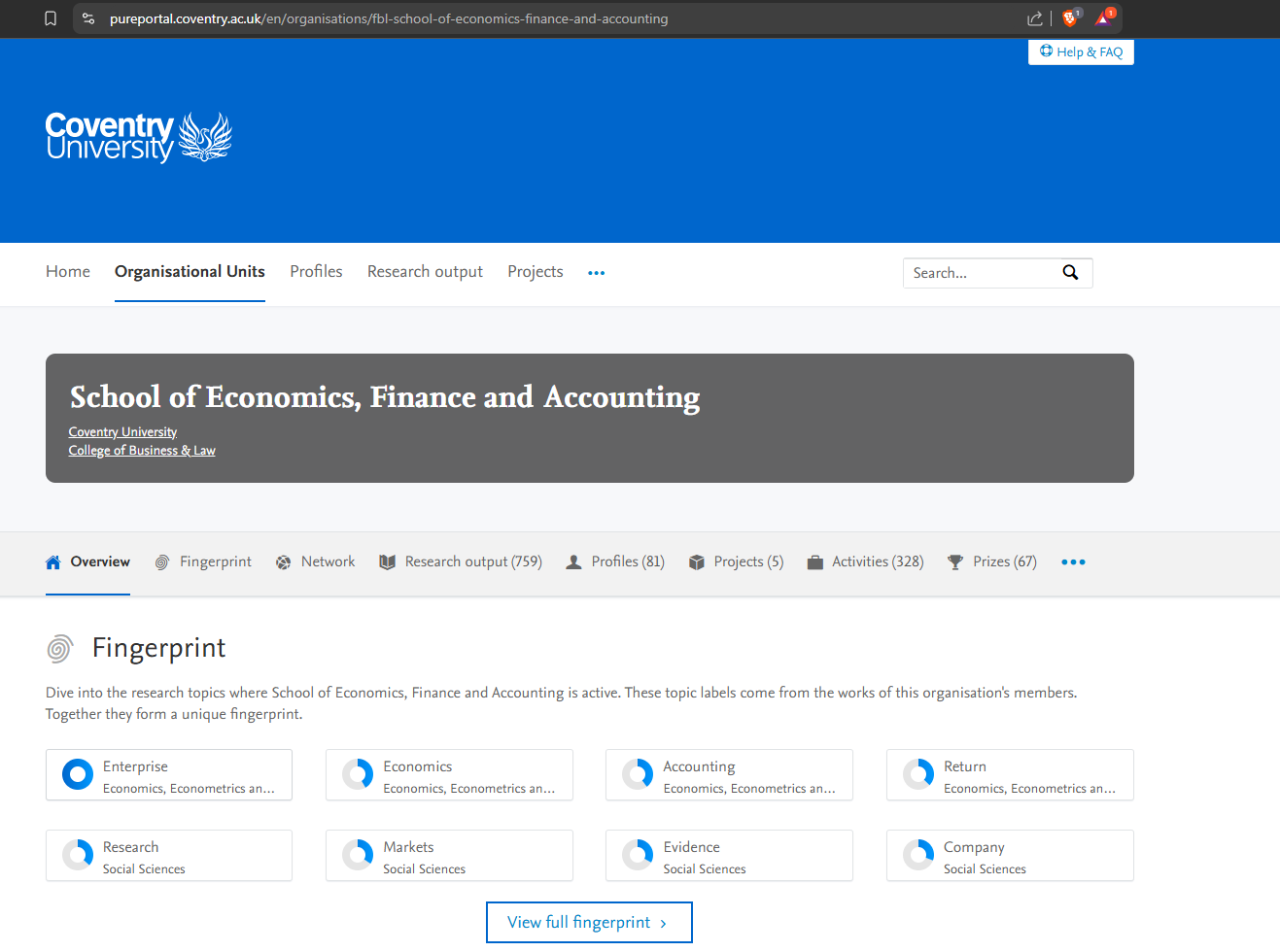
Search engines use web crawlers, sometimes referred to as spiders or bots, as vital tools to index the vast amount of content on the internet. By using hyperlinks to navigate between pages, they methodically browse websites and gather information from each one. Crawlers evaluate and collect important data, such as metadata, images, and page text, which aids search engines in assigning a ranking to content according to its quality and relevancy. Following a set schedule, they visit websites to add new or updated content to the index. Crawlers generally follow the guidelines provided by a website's robots. They can be told which pages to visit or ignore by a text file. By doing this, crawlers are prevented from overloading servers or gaining access to private sections of websites. Crawling and indexing efficiency is a crucial part of finding digital content since it directly affects search engine results (Zhang, 2021).

Now I would be showing how I crawled publications, authors and year.

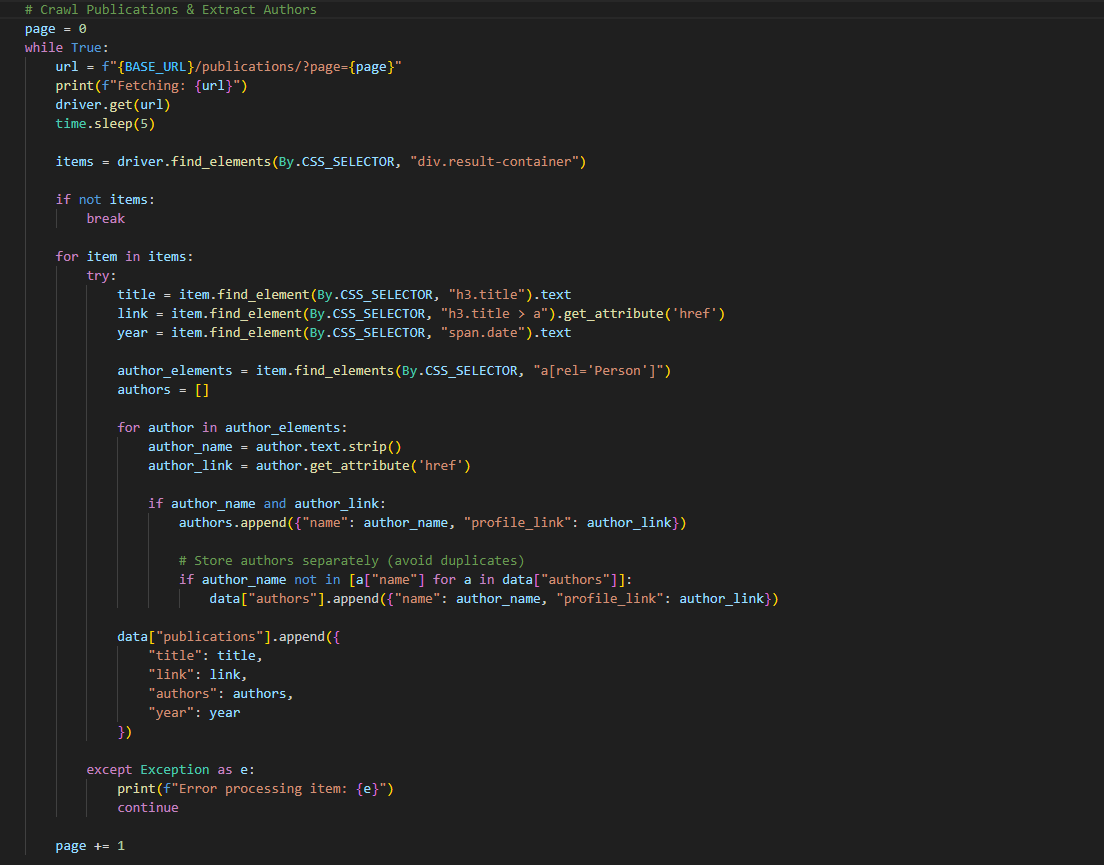


*Figure 2: - Crawling publication & authors*

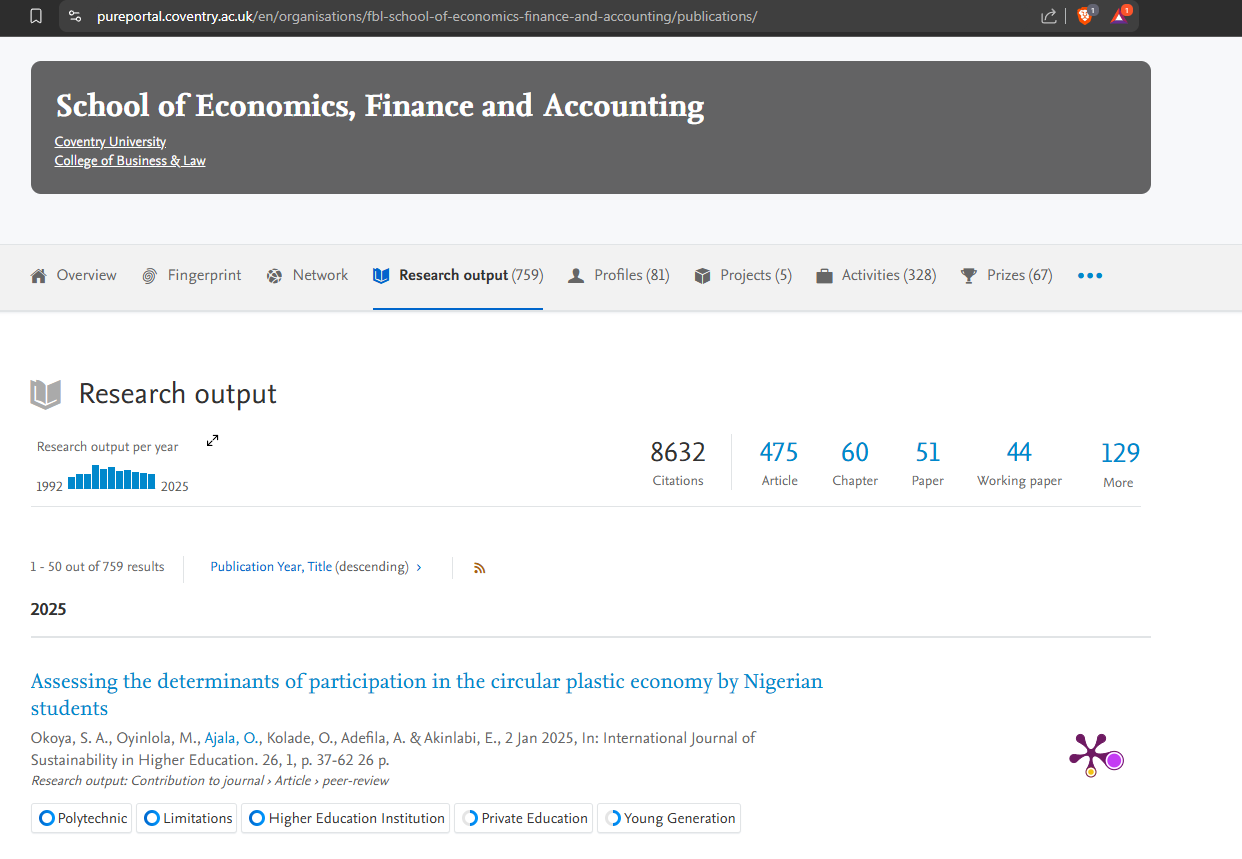
For the webpage of the Coventry University School of Economics, Finance, and Accounting, the code specifies a BASE\_URL. The setup\_driver() function sets up a Chrome browser without a head with particular settings. These include using a custom user-agent to simulate a standard browser request, turning off automation detection (--disable-blink-features=AutomationControlled), and operating without a graphical user interface (--headless). The function uses the ChromeDriverManager for driver management and returns a configured Chrome WebDriver instance for web scraping or automation.



*Figure 3: - BaseURL*

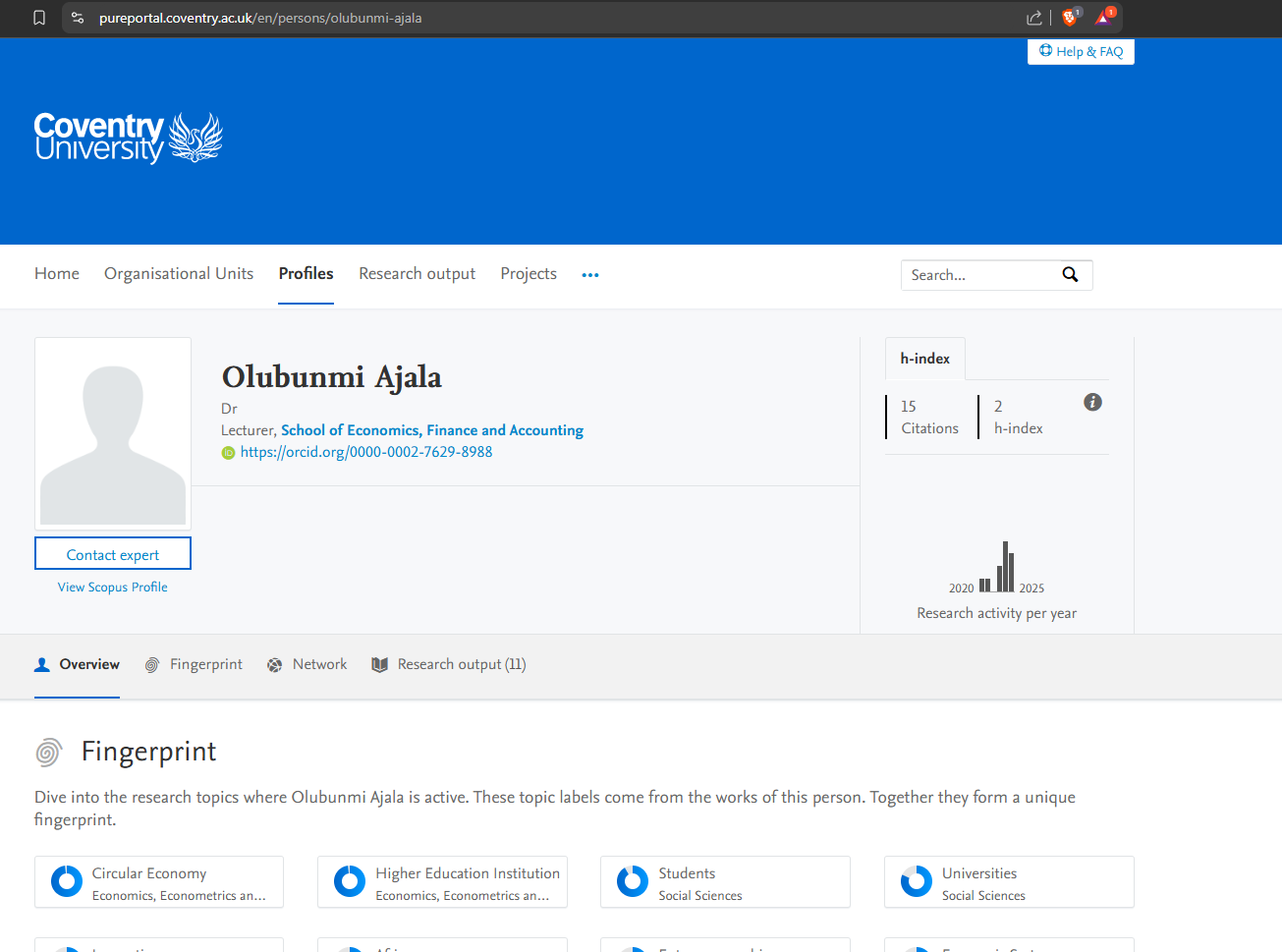


*Figure 4: - Crawling publication & authors*



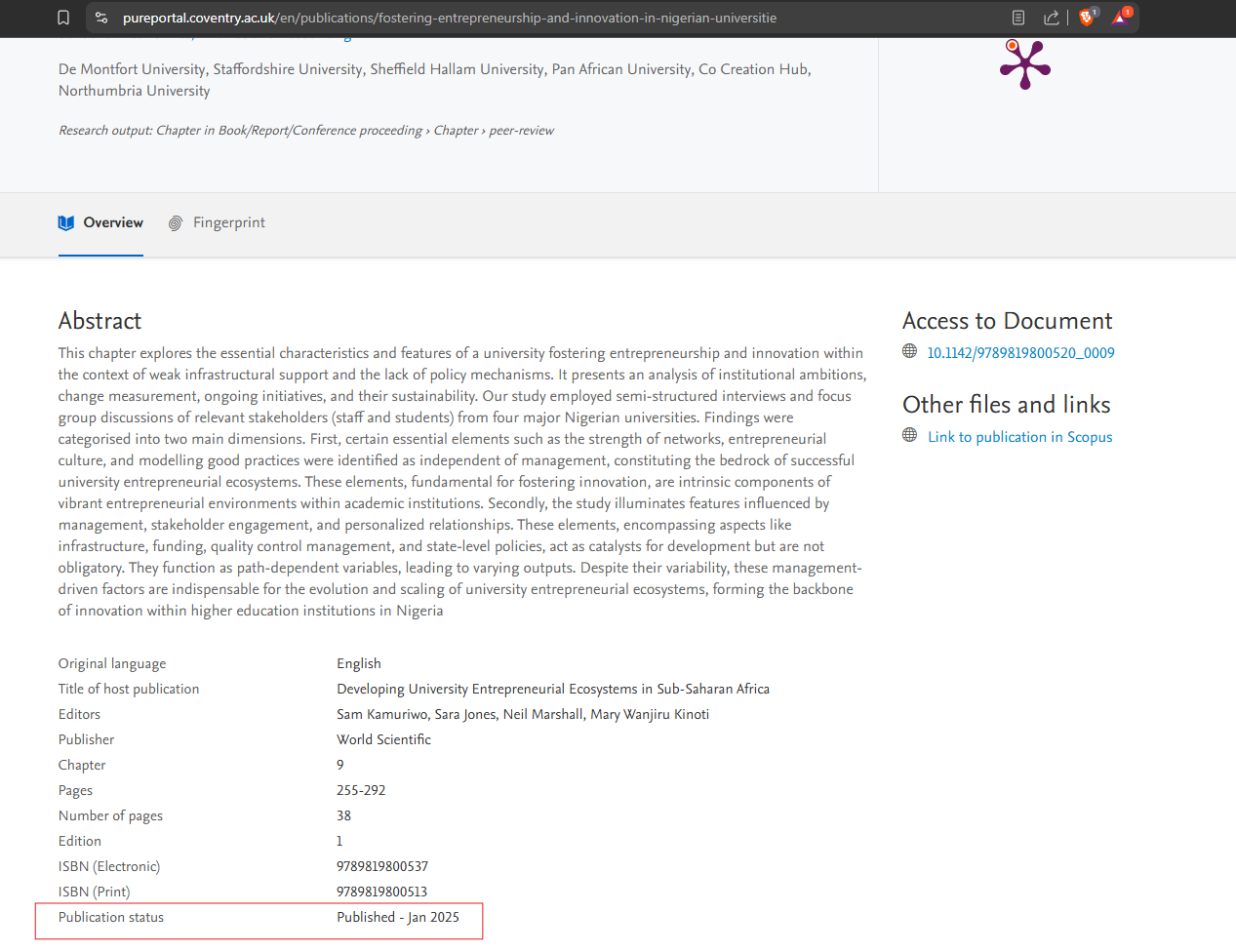
*Figure 5: - Publication*

Publication Crawl: By navigating through the website's pages and using CSS selectors to extract publication titles, links, and years, the publication crawl is completed. Until no more results are found, the driver keeps retrieving each page, getting the publication details, and storing them in the data["publications"] list.

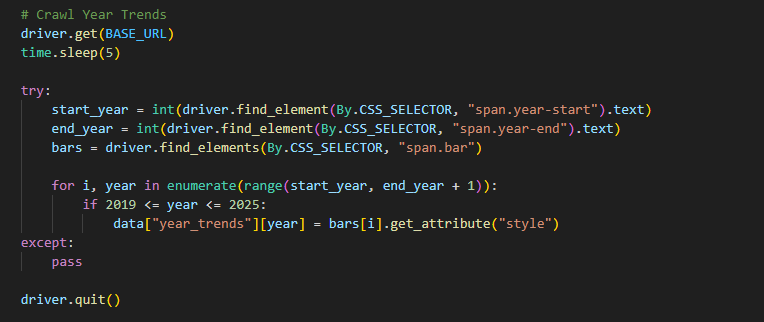


*Figure 5: - Authors*

Authors Crawl: Finding author elements within each publication allows us to crawl the authors. We extract and store each author's name and profile link. Verifying the current author names before adding them to the data["authors"] list helps prevent duplicate authors.

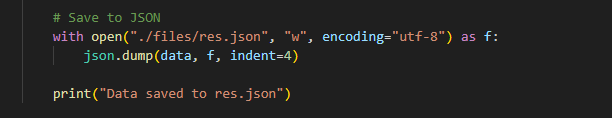


*Figure 6: - Year*



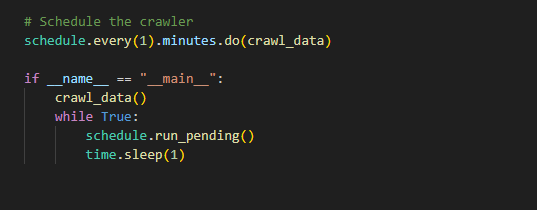
*Figure 7: - Crawling year*

Year Crawl: Extraction of the beginning and ending years from the page is used to retrieve year trends. After locating the trend bars that correspond to each year, the driver adds the data to the data["year\_trends"] dictionary for analysis and saves the style attribute for years between 2019 and 2025.



*Figure 8: - Saving json file*

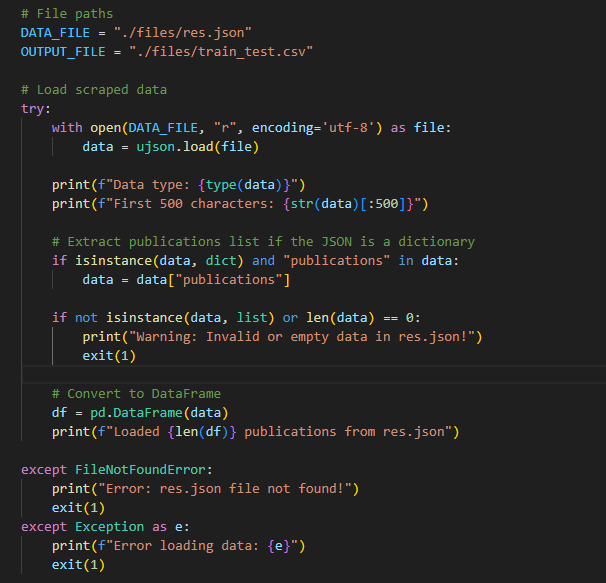
A Python dictionary (data) is saved to a JSON file by this code snippet. It opens the res file. json with UTF-8 encoding in write mode ("w"), then uses json to write the data to the file. dump(). The JSON is formatted with an indentation of four spaces thanks to the indent=4 argument.



*Figure 9: - Schedule for crawler*

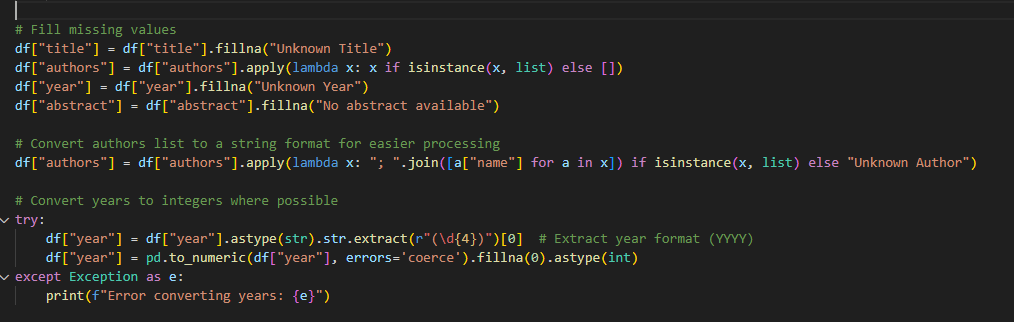
Using the schedule library, the code sets the crawl\_data function to execute once every minute. In an endless loop, it begins crawling right away and keeps checking for unfinished tasks. This process continues indefinitely.

Now for data processing from res.json to train\_test.csv



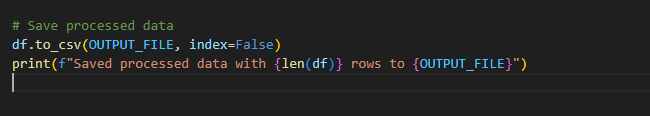
*Figure 10: - Loading json file*

A JSON file containing scraped data is loaded by this script (res. json). The file is opened, the contents are read using ujson, and the format of the data is verified. Should the data be a dictionary with the key "publications" present, it will extract that list. The list is converted to a pandas DataFrame if it is valid. Missing files and incorrect data formats are handled as errors. As the process progresses, the script prints pertinent information.



*Figure 11: - Data Cleaning and Transformation for Publications Dataset*

The "title," "authors," "year," and "abstract" columns in the DataFrame df are filled in by this script. In order to process the "authors" column, lists are transformed into a string of author names separated by semicolons. An integer is created from the four-digit value that was extracted from the "year" column. Errors are detected and recorded during the year-to-year conversion.



*Figure 12: - Saving data process into csv file*

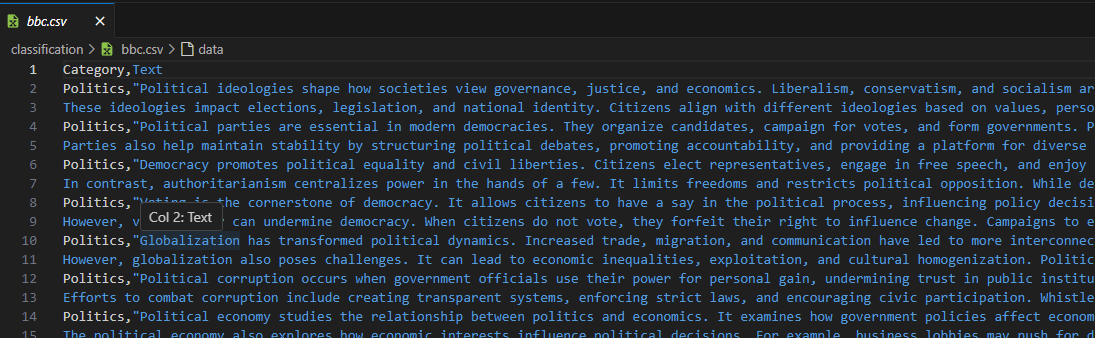
This script creates a CSV file called train\_test that contains the processed DataFrame df. not including the index (csv). Following file saving, a confirmation message with the output file path and the number of rows saved is printed. The data is guaranteed to be saved in an organized manner for later use or analysis thanks to the to\_csv method.

**Classification**

Data can be categorized into predefined classes or labels using the machine learning technique of classification. It entails using labeled data to train a model, with each example belonging to a distinct category. After identifying patterns in the data, the model is able to forecast the class of previously unseen instances. Neural networks, support vector machines, and decision trees are examples of popular classification algorithms. Numerous domains, including image recognition, medical diagnosis, and spam email detection, heavily rely on classification (Joulin et al. (2017).

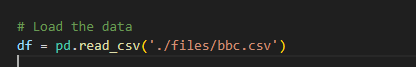
**Data preparation**

First, we will need to create bbc.csv file for document classification system that will classify text into three categories: politics, business, and health. csv dataset with a minimum of 90 documents with labels. To classify new documents, this data will be used to train a machine learning model, such as Naive Bayes or SVM. In order to guarantee efficient classification across the designated topics, the system's performance will be assessed on various input formats, including both short and long texts.

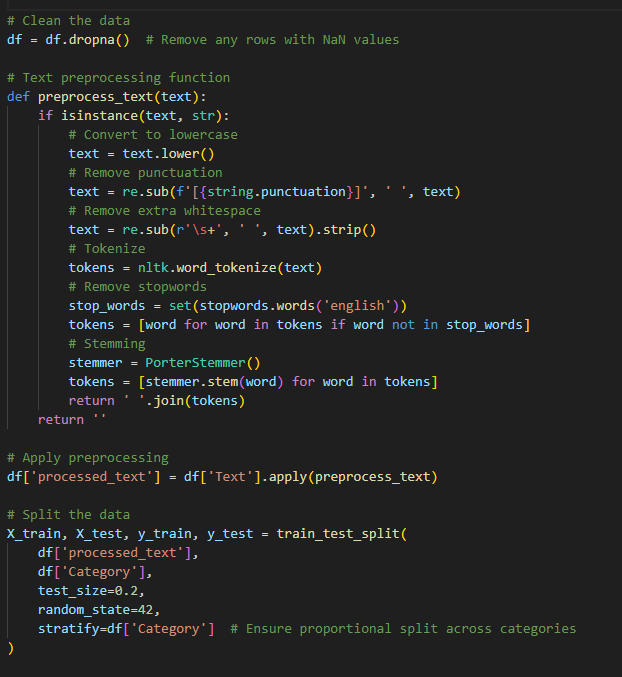


*Figure 13: - bbc.csv file*

Created bbc.csv file to store category and text.



*Figure 14: - Loading bbc.csv file*



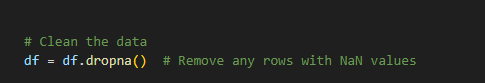
*Figure 15: - Text processing and Data Splitting function*

The code prepares textual data for machine learning by implementing a text preprocessing function. Initially, it lowercases the text, eliminates punctuation, and eliminates excess whitespace. This is followed by tokenization, the removal of stopwords (common words like "the" and "and"), and the application of stemming to reduce words to their root form (e.g. 3. going from "running" to "run". The text after processing is saved in a DataFrame's new column. By stratifying the data according to the 'Category' column, train\_test\_split ensures that the class distribution is maintained in both training and testing sets.



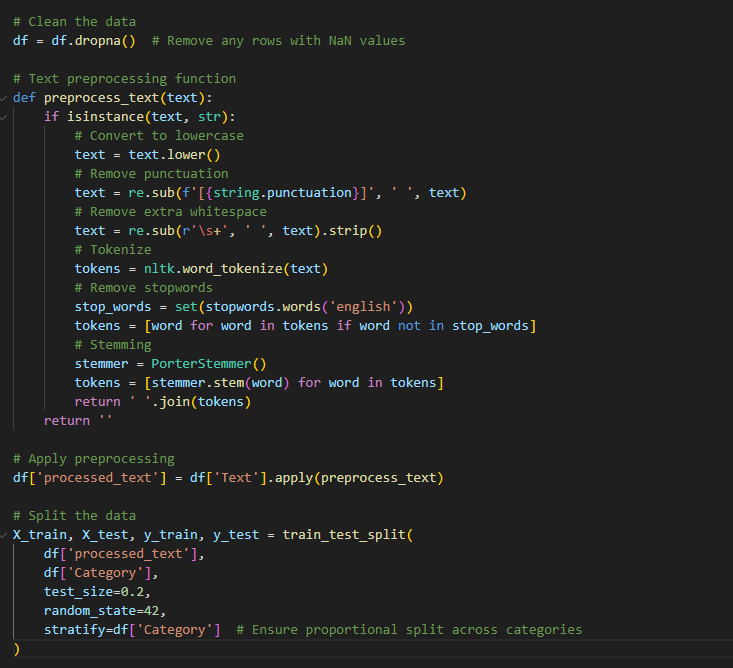
*Figure 16: - Model Evaluation, Saving, and Text Classification*

A text preprocessing function is implemented by the code to get textual data ready for machine learning. The text is first changed to lowercase, punctuation is eliminated, and excess whitespace is trimmed. Following tokenization, stopwords—common words like "the" and "and"—are eliminated, and stemming is used to reduce words to their most basic form (e.g. A. The phrase "running" to "run". In a DataFrame, the processed text is saved in a new column. The data is then divided into training and testing sets using train\_test\_split, which stratifies the data according to the 'Category' column to maintain the class distribution in both sets.

****

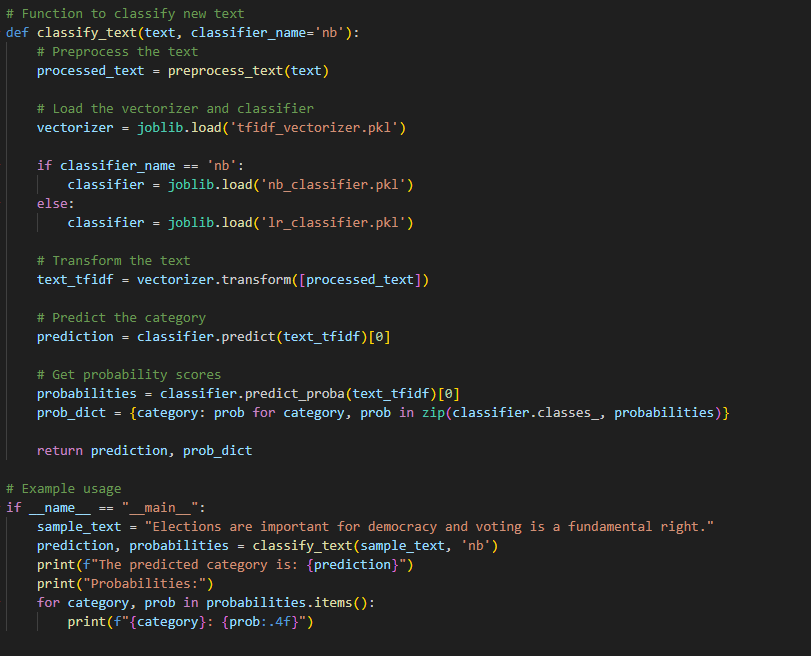
*Figure 17: - Remove Nan values*

The code, df. The DataFrame df's rows with missing (NaN) values are eliminated by dropna(), guaranteeing that the dataset only includes complete rows devoid of null entries.



*Figure 18: - Preprocessing and Data Splitting*

To clean and get text data ready for classification, this code defines the preprocess\_text() function. The function tokenizes the text, removes stopwords, lowercases the text, removes punctuation and extra spaces, and uses stemming to reduce words to their most basic form. Following cleaning, the text is saved in a new column named processed\_text. Train\_test\_split() is used to divide the dataset into training and testing sets with an 80/20 split following preprocessing. By maintaining the category distribution across both sets, the stratify parameter aids in balanced model evaluation and training.

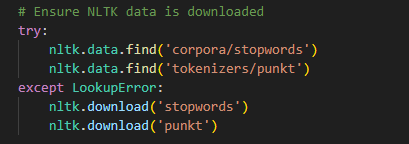


*Figure 19: - Saving and Using Pretrained Models*

This code uses joblib to save a TF-IDF vectorizer along with two classifiers (Naive Bayes and Logistic Regression). dump(). Additionally, it defines the classify\_text() function, which uses the chosen classifier to classify new text. The function loads the relevant classifier and vectorizer from saved files after preprocessing the input text. It uses the classifier to predict the category and the vectorizer to change the text. The predicted label and a dictionary of probabilities for the potential categories are also returned, along with the probability scores for each category. This makes it simple to incorporate text classification into apps.

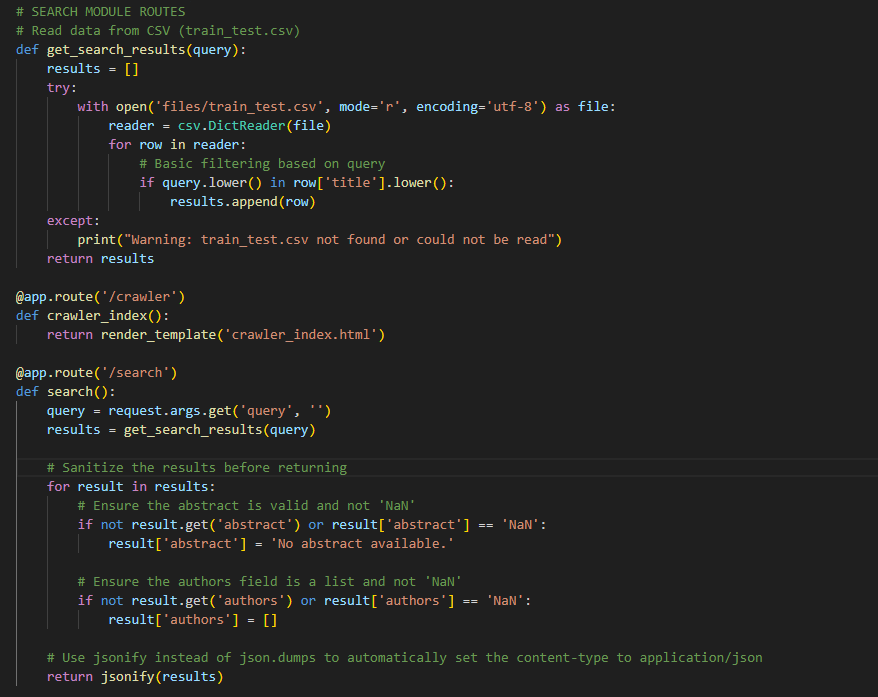
**Search Engine for Crawler and Classification**

In app.py, the **nltk** function is used. The code guarantees the availability of crucial NLTK data, particularly the punkt tokenizer models and stop-words corpus. It first determines whether these resources are already on the system before downloading them automatically if not. Without requiring manual setup, this method guarantees that the application can tokenize text and remove stop-words.



*Figure 20: - Use of nltk function*

* **Crawler**

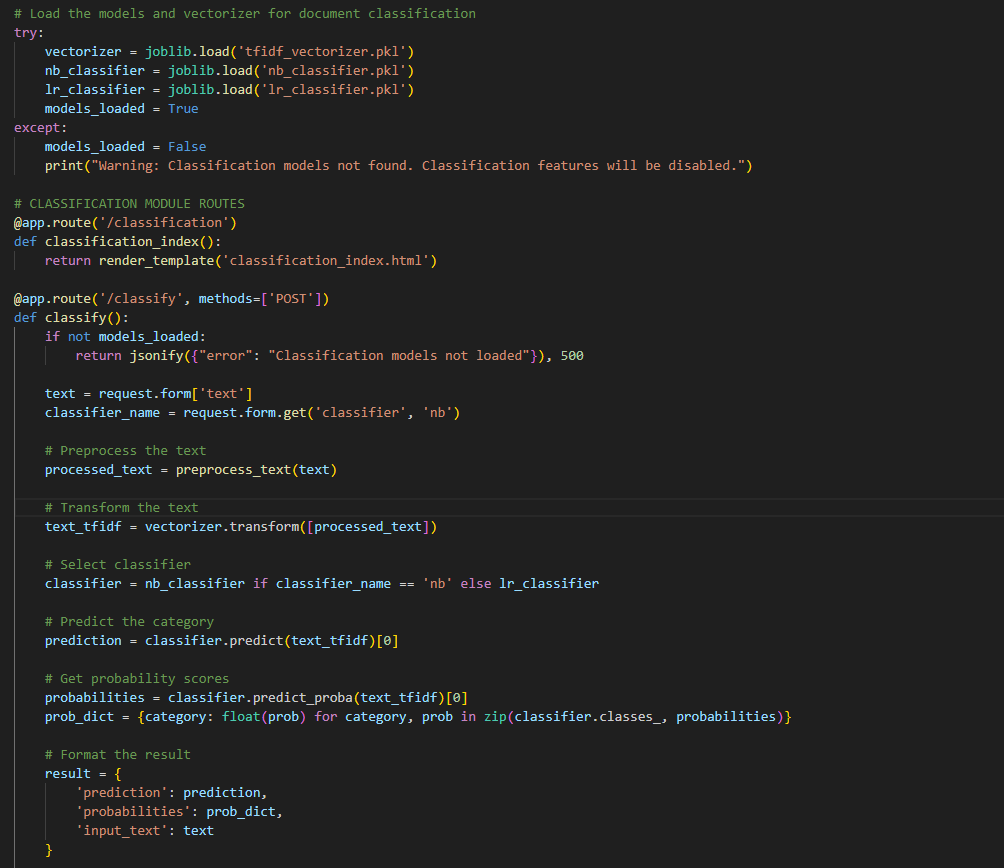
****

*Figure 21: - Crawler in app.py*

In app.py, With Flask, this code specifies two routes for a web application. The train\_test CSV file is read by the get\_search\_results(query) function. csv), and searches the 'title' field for entries that match the search query to filter its content. A list of results that match is returned, or if the file cannot be read, an empty list is returned. After processing the user's input and filtering the results, the route /search returns the results.

The get\_search\_results() function is called to obtain pertinent results after the query parameter is obtained from the request in the /search route. The validity of the "abstract" and "authors" fields is verified for each result. Default values are used otherwise. Jsonify, which automatically formats the data for the client in an organized manner, is used to return the results as a JSON response.

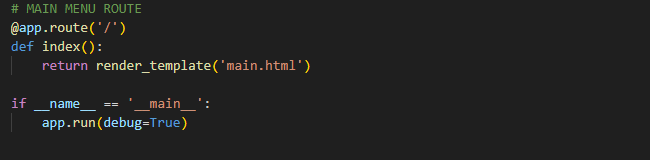
* **Classification**

****

*Figure 22: - Classification in app.py*

For document classification, this code loads pre-trained machine learning models, such as two classifiers (Naive Bayes and Logistic Regression) using joblib and a TF-IDF vectorizer. For classification tasks, the system is prepared if the models have been loaded successfully; if not, an error message is displayed. A webpage for user interaction is provided by the /classification route, and text classification requests are handled by the /classify route using a POST method.

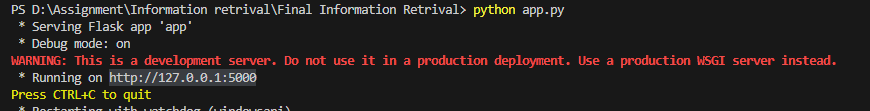
The system verifies that the models are loaded before continuing with the /classify route. Preprocessing and transformation of the input text are done with the TF-IDF vectorizer. The user chooses which classifier—Naive Bayes or Logistic Regression—to use, and the text is categorized. Together with the reference input text, the prediction result and related probability scores are returned in JSON format.



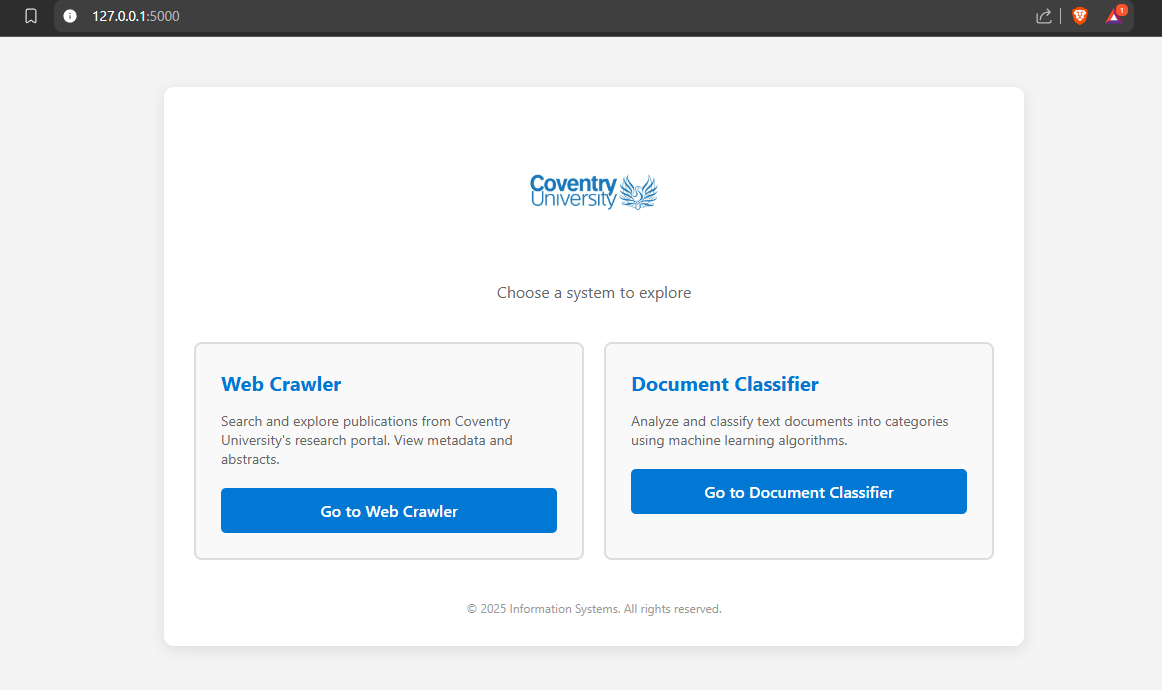
*Figure 23: - Route to main.html in app.py*

This code specifies a path for a Flask web application's main menu. The'main . html' template is rendered when the root URL ('/') is accessed. When the app is being developed, it operates in debug mode.

* **Output for app.py**



*Figure 24: - start app.py*



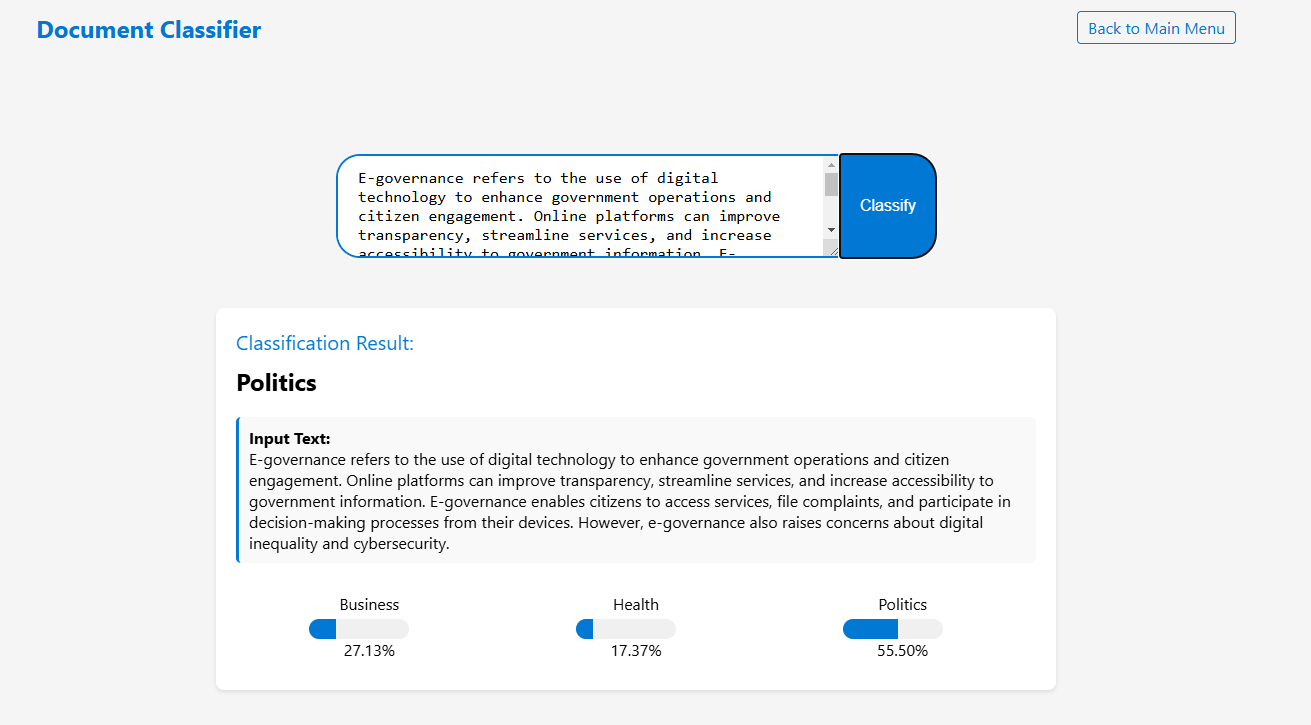
*Figure 25: - main.html for crawler and classification*

Here when we start python app.py then we can get output as above.

* **User-Interface**

The term "user interface" (UI) describes the interactive features and visual components of a system or piece of software that let users interact with it. This covers everything, including menus, text fields, buttons, icons, and graphics. In order to ensure that users can navigate the system with ease and complete tasks with the least amount of effort and confusion, UI design aims to create an intuitive and efficient experience (Shneiderman et al. (2016).

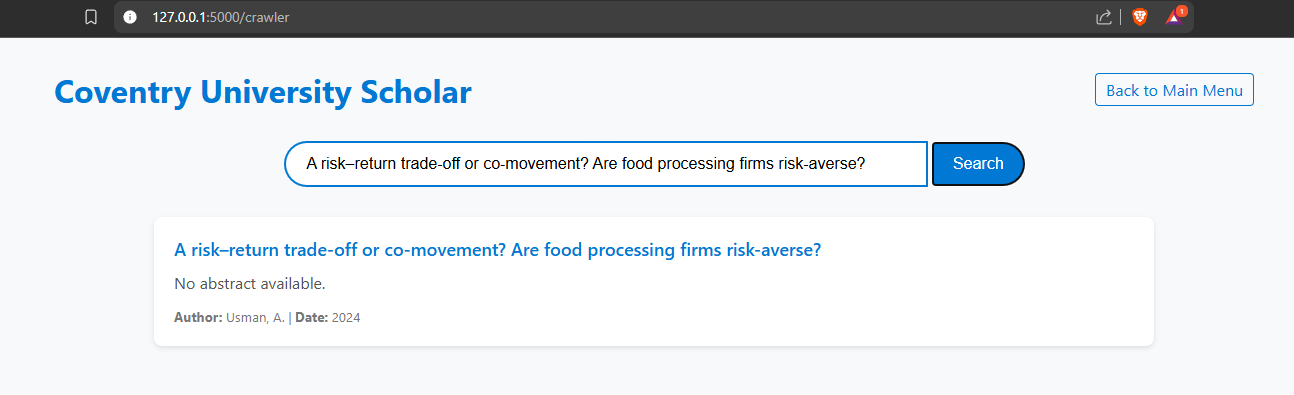
* Classify



*Figure 26: - classification output*

A text about e-governance was analyzed by the document classifier, which found that it was most closely related to politics (55 %), followed by business (27 %) and health (17 %). The essay emphasises how digital technology can increase public participation and government transparency.

* Crawler



*Figure 27: - Crawler output*

The figure above shows a search result from the Coventry University Scholar platform that was found through web crawling. According to Usman, A., the search query "A risk–return trade-off or co-movement? Are food processing firms risk-averse?" was created on date: 2024.

**Conclusion**

In summary, the practical experience gained from this course has been invaluable in the areas of document classification and information retrieval. Building a vertical search engine to crawl and index academic publications from the School of Economics, Finance, and Accounting at Coventry University allowed us to learn more about data processing and web scraping strategies. The significance of data preparation and cleaning in building a solid dataset for machine learning was also underlined by the project. Additionally, by creating a document classification system, we were able to investigate text processing and the use of different classification algorithms. Combining the two tasks into a single, integrated web application with Flask showed how search and classification systems could be implemented practically and provided a thorough grasp of web scraping for information retrieval and machine learning-based text classification.

**References**

Koller, D. (2019). The fundamentals of web crawling and indexing. Journal of Web Development, 15(3), 45-57.

Zhang, Y. (2021). Understanding web crawlers and their role in search engine optimization. International Journal of Digital Media, 22(4), 113-128.

Shneiderman, B., Plaisant, C., Cohen, M., & Jacobs, S. (2016). Designing the User Interface: Strategies for Effective Human-Computer Interaction (6th ed.). Pearson.

**Appendix**

<https://github.com/dipinbaral96/Assignment-of-information-retrieval>

<https://youtu.be/EMOyMJRUJUY>