Contents

[Task 1: Search Engine 2](#_Toc193660054)

[**1. Introduction** 2](#_Toc193660055)

[2. Web Crawler (crawl.py) 2](#_Toc193660056)

[a) Importing necessary libraries 2](#_Toc193660057)

[b) Extracting Publication Data 2](#_Toc193660058)

[c) Crawling Multiple Pages 4](#_Toc193660059)

[4. Search Interface (search.py) 5](#_Toc193660060)

[a) Importing required Libraries and CSV file 5](#_Toc193660061)

[b) Creating Tiknter GUI 6](#_Toc193660062)

[c) Search Functionality 6](#_Toc193660063)

[d) Clickable Links in Search Results 7](#_Toc193660064)

[5. Scheduling with Task Scheduler 8](#_Toc193660065)

[6. Conclusion 9](#_Toc193660066)

[Task 2: Subject Classification 10](#_Toc193660067)

[**1.** **Introduction** 10](#_Toc193660068)

[**2.** **Data Collection** 10](#_Toc193660069)

[**3.** **Preprocessing** 10](#_Toc193660070)

[4. Model Selection 12](#_Toc193660071)

[i. Logistic Regression 12](#_Toc193660072)

[ii. Navie Bayes Model 13](#_Toc193660073)

[5. Document Classification System 14](#_Toc193660074)

[6. Conclusion 15](#_Toc193660075)

[References 16](#_Toc193660076)

[Figure 1 Importing necessary libraries 3](#_Toc193740778)

[Figure 2 Extracting Publication Data 4](#_Toc193740779)

[Figure 3 Extracting and Navigating Next Page URL 4](#_Toc193740780)

[Figure 4 Structured Data Extraction from Webpage 5](#_Toc193740781)

[Figure 5 Structured Data Extraction from Webpage 6](#_Toc193740782)

[Figure 6 Importing required Libraries and CSV file 7](#_Toc193740783)

[Figure 7 Creating GUI 7](#_Toc193740784)

[Figure 8 Search Functionality 8](#_Toc193740785)

[Figure 9 Clickable Links in Search Results 8](#_Toc193740786)

[Figure 10 Scheduling with Task Scheduler 10](#_Toc193740787)

[Figure 1: Importing Data Set 11](#_Toc193740788)

[Figure 2 Preprocessing the Data Set 12](#_Toc193740789)

[Figure 3: Splitting Data Set 12](#_Toc193740790)

[Figure 4: Logistic Regression 13](#_Toc193740791)

[Figure 5 Document Classification Logistic Regression 13](#_Toc193740792)

[Figure 6 Naive Bayes Model 14](#_Toc193740793)

[Figure 7 Document Classification Naive Bayes 14](#_Toc193740794)

[Figure 8 Document Classification System 15](#_Toc193740795)

# Task 1: Search Engine

**1**. Introduction

Vertical search is a term used to describe niche or specialty search engines that focus on a single industry vertical, type of content - or a specific segment of the overall search (Shettar & Bhuptani, 2007). The aim of this task is to create a vertical search engine designed to retrieve academic publications by members of Coventry University’s School of Economics, Finance, and Accounting. We built a vertical search engine by crawling relevant web pages and extracting required information.

## 2. Web Crawler (crawl.py)

A web crawler, is a program used to systematically browse the internet and collect information from the web pages.

### a) Importing necessary libraries

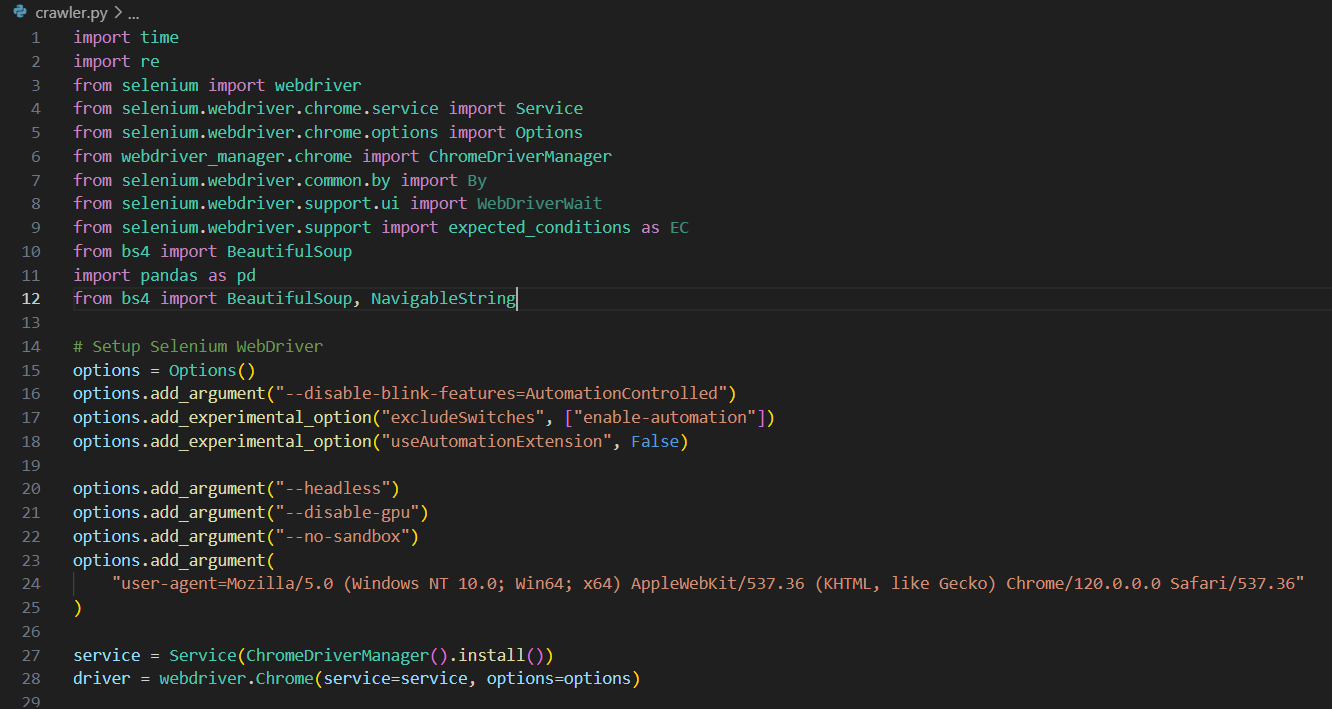


Figure 1 Importing necessary libraries

The initial step began with importing all the necessary libraries. Those libraries include Selenium for web automation, BeautifulSoup for HTML parsing, and Pandas for providing data structures and data representation. It also installs and initializes ChromeDriver. All this installation of the necessary libraries and drivers are done to optimize web scrapping tasks.

### b) Extracting Publication Data

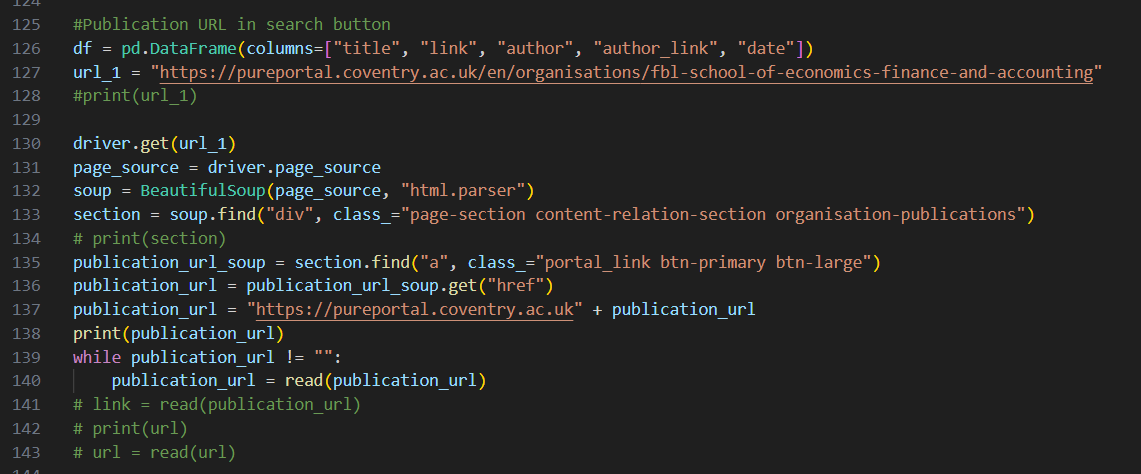


Figure 2 Extracting Publication Data

The above python script focuses on scrapping publication URLs from the Coventry University’s Pure Portal for school of economics, finance and accounting. The code navigates to the given URL and retrieves the HTML content of the page. The script extracts publication URLs from the webpage and iterates through multiple pages to ensure all available publication links are captured successfully.

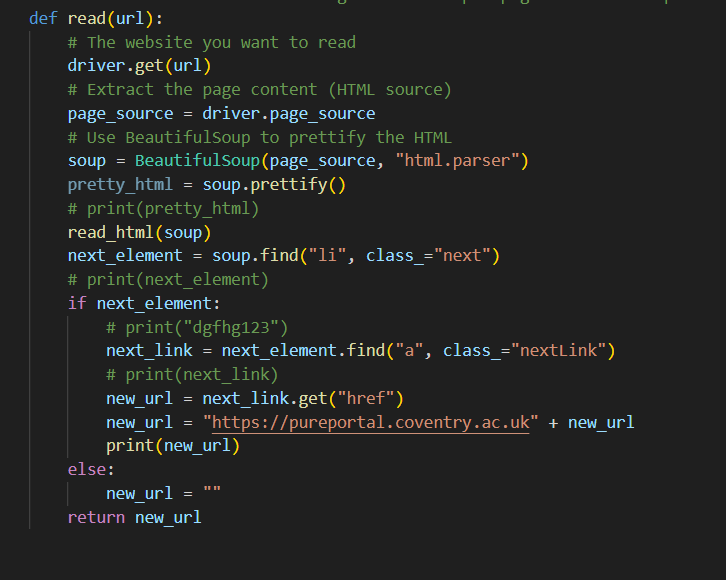


Figure 3 Extracting and Navigating Next Page URL

The above python code automates the process of navigating the web pages and extracting the URL of the next page. A function is used to search for an element with a specific class, which is assumed to contain the link to the subsequent page. If found, it extracts the attribute from the tag and constructs the full URL. If no such element is found, the function assigns an empty string.

### c) Crawling Multiple Pages

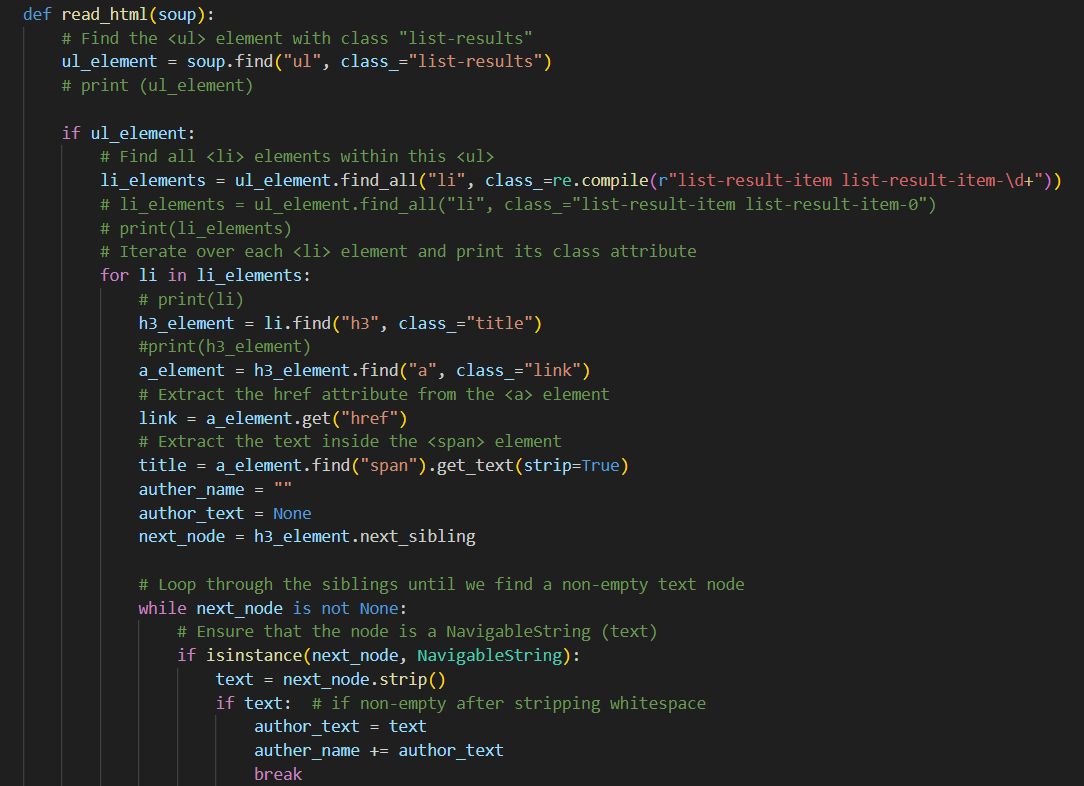


Figure 4 Structured Data Extraction from Webpage

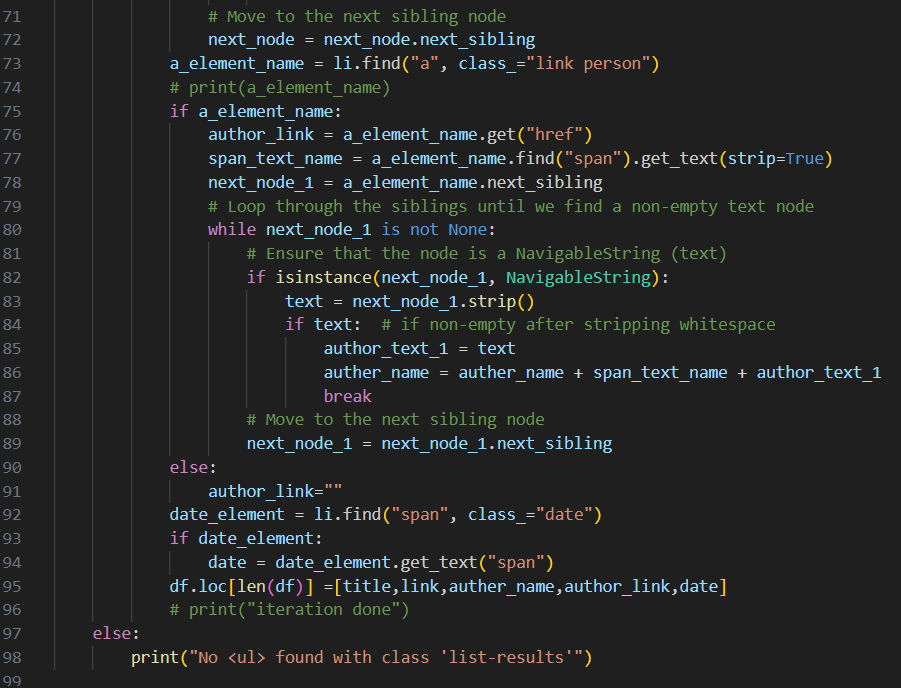


Figure 5 Structured Data Extraction from Webpage

The above python script extracts structured data from the list of results. It retrieves key details such as title, link, author name, author link, and date. It first extracts the title and link from heading elements. It then locates a link with the class to extract author’s link and name. It further concatenates the full author’s name if needed. If the author’s link is not found, it is set to an empty string. Subsequently, the function extracts the date from a span with the specified class. Finally appends the title, link, author name, author link, and date as a new row to a DataFrame. If the initial list container is not found, it prints an error message. Overall, this function retrieves title, link, author, and date information from a list of results within the HTML. The retrieve dataset is saved as an csv.

## 4. Search Interface (search.py)

This section enables users to search for and filter specific data, such as publications, within a dataset. It allows users to search term, process the input, and display relevant results based on matching criteria. It also provides functionality to make URLs within the results clickable, so users can easily navigate to the web addresses directly from the table.

### Importing required Libraries and CSV file

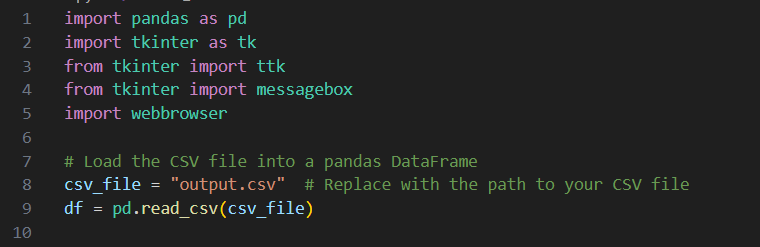


Figure 6 Importing required Libraries and CSV file

### b) Creating Tiknter GUI

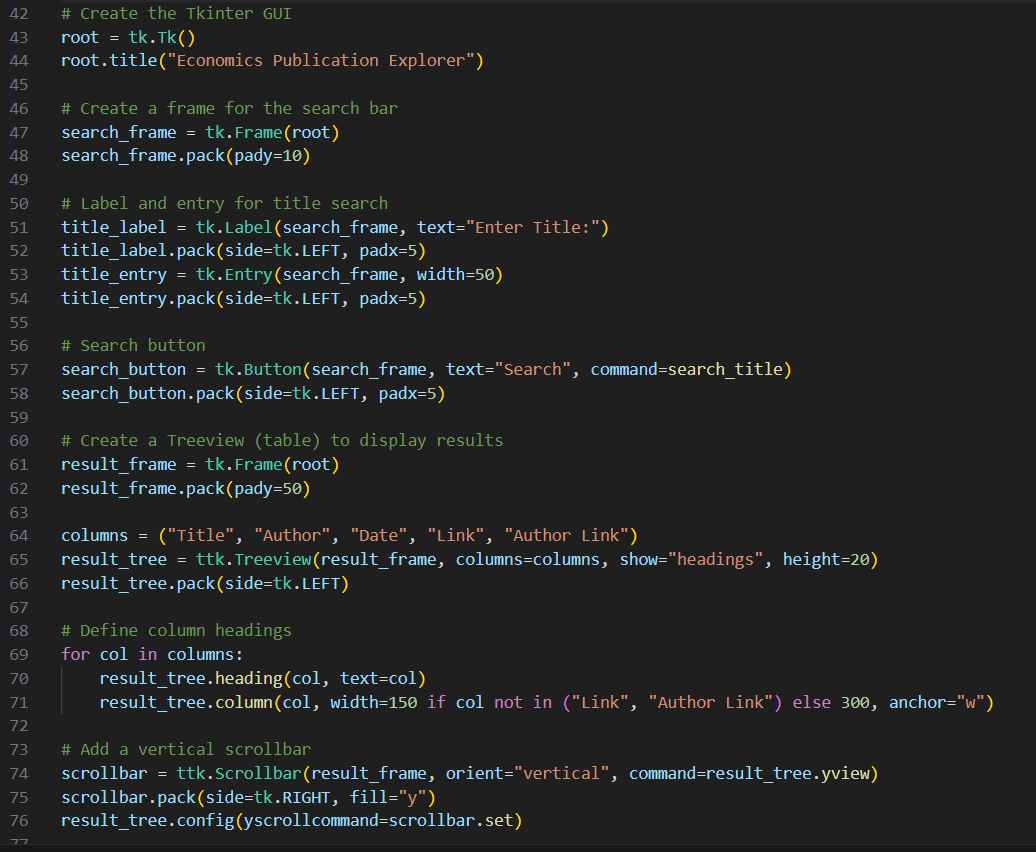


Figure 7 Creating GUI

The above code is for creating simple GUI application using Tkinter for searching and displaying all the publication extracted. It set up a window with a search bar for titles, a search button, and a table to show results. The table has column for title, author, date, link, and author link. A scrollbar is added for longer result lists.

### c) Search Functionality



Figure 8 Search Functionality

The search\_tittle function clears any previous results from the table and gets the search term entered by the user. It then checks if the user has entered a term, and if not, it shows an error message. It search through the dataset to find rows where the title column contains the search term. If there are matches, it adds the results to the table and displays the title, author, date, link and author link for each publication.

### d) Clickable Links in Search Results

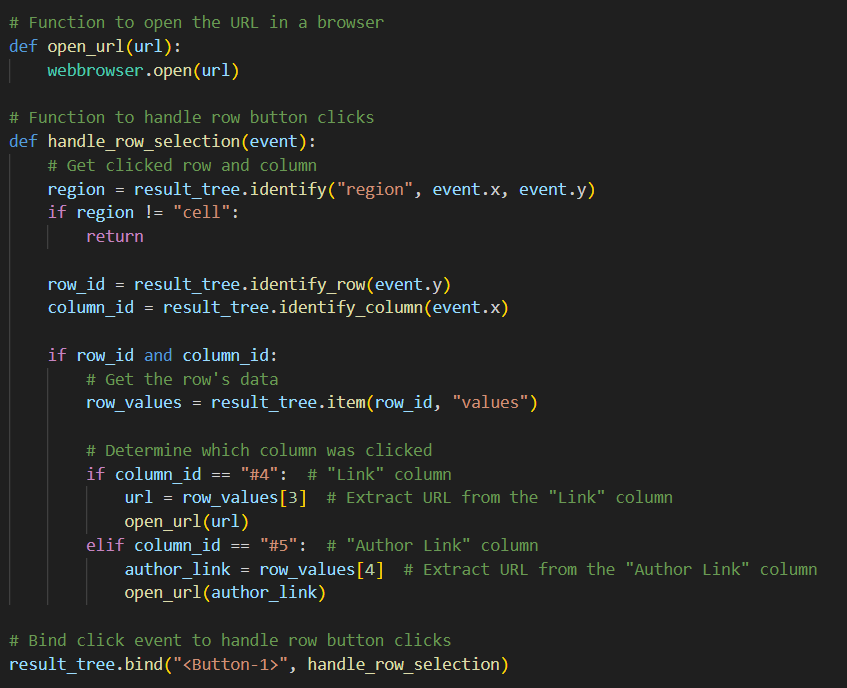


Figure 9 Clickable Links in Search Results

The python code makes the given URLs in a clickable table. When a user clicks on the cell, the given function checks which cell was clicked. If the clicked cell is in the “Link” or “Author Link” columns, it opens the URL in a web browser. In short, this code is prepared to make clickable URL that opens directly in a browser.

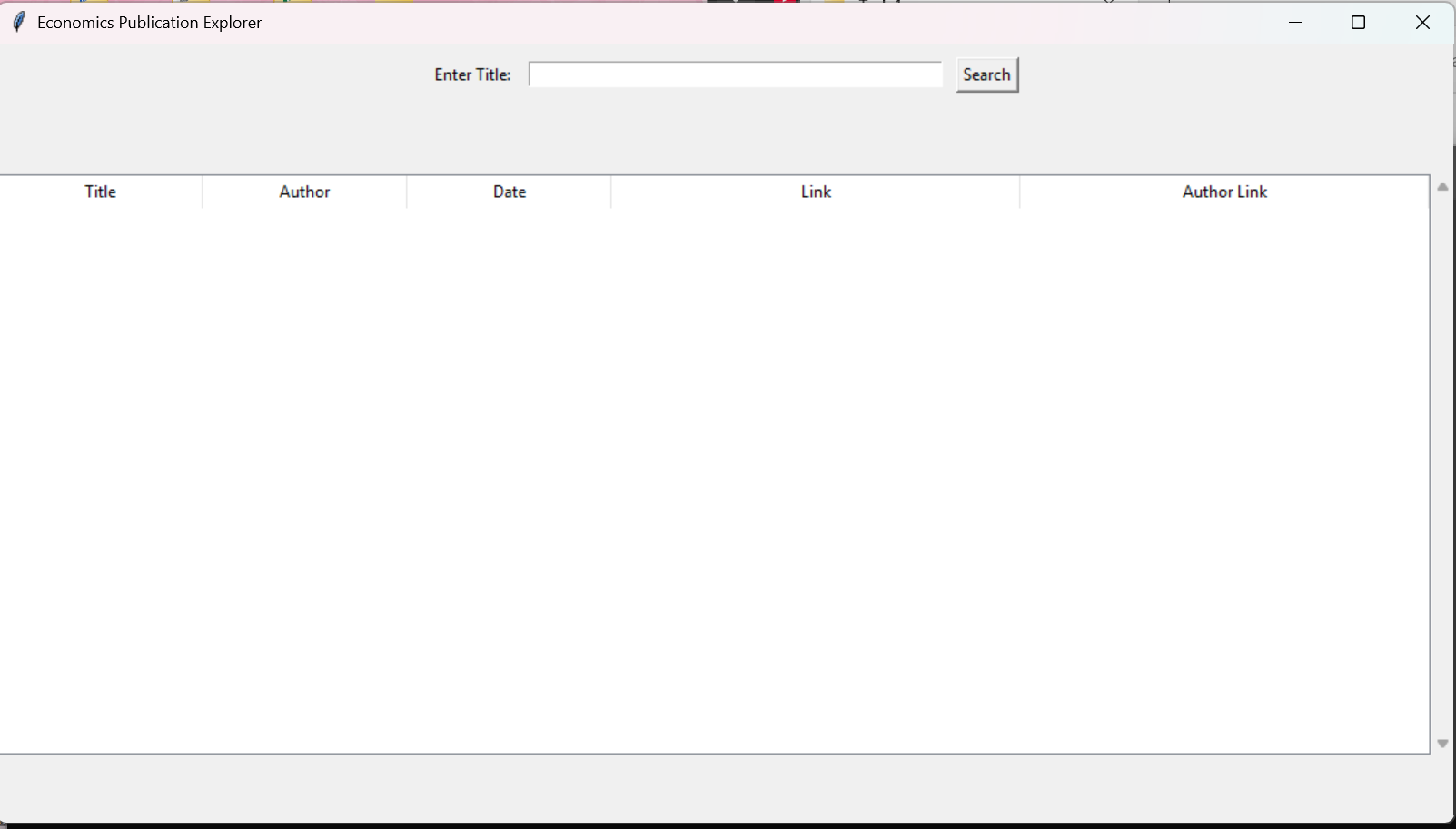
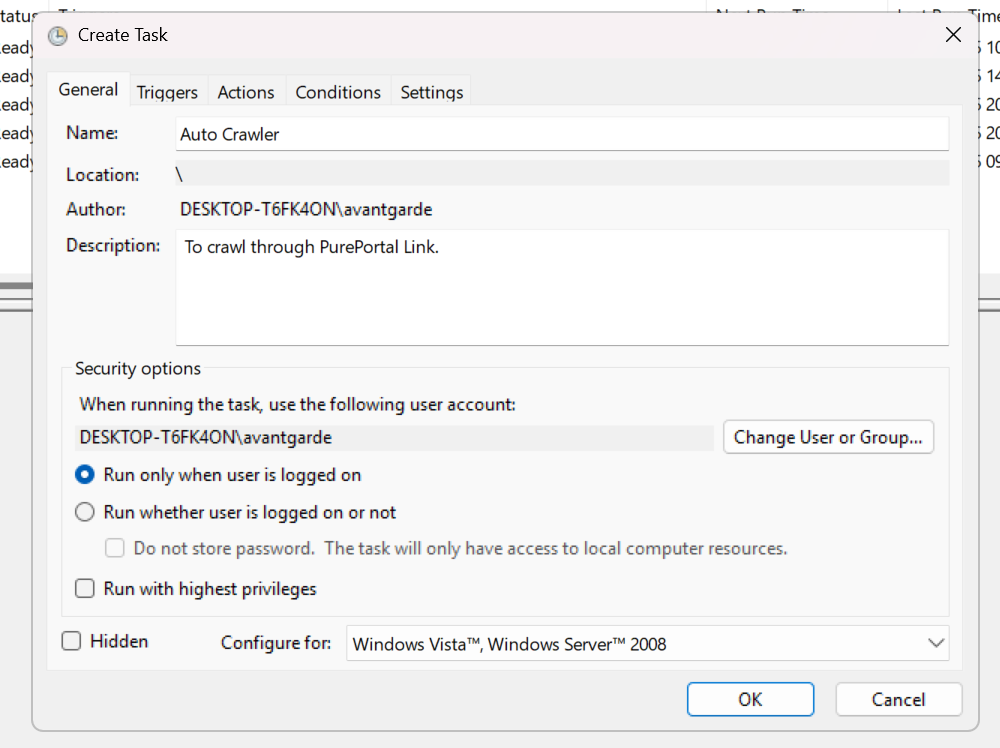
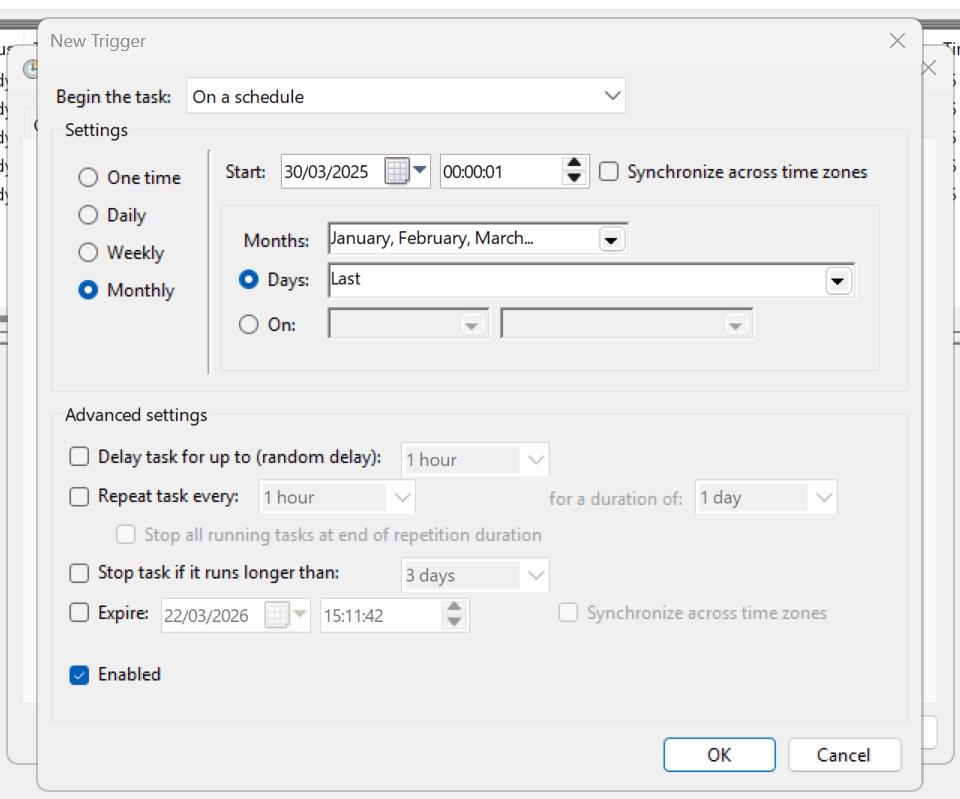
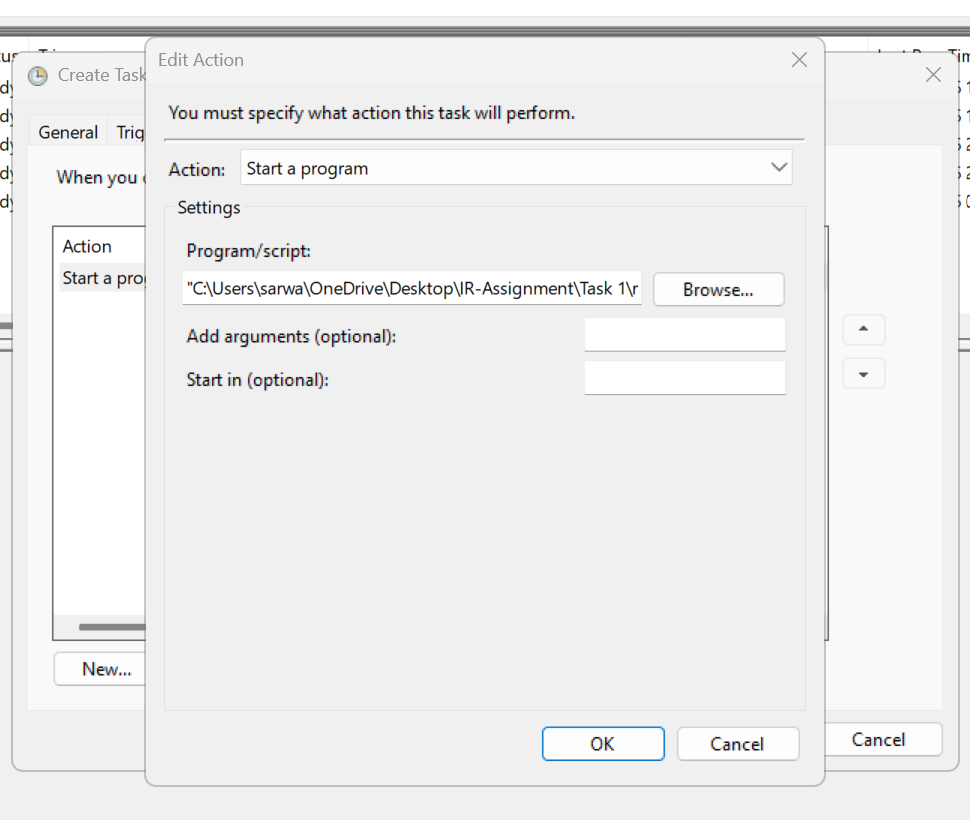
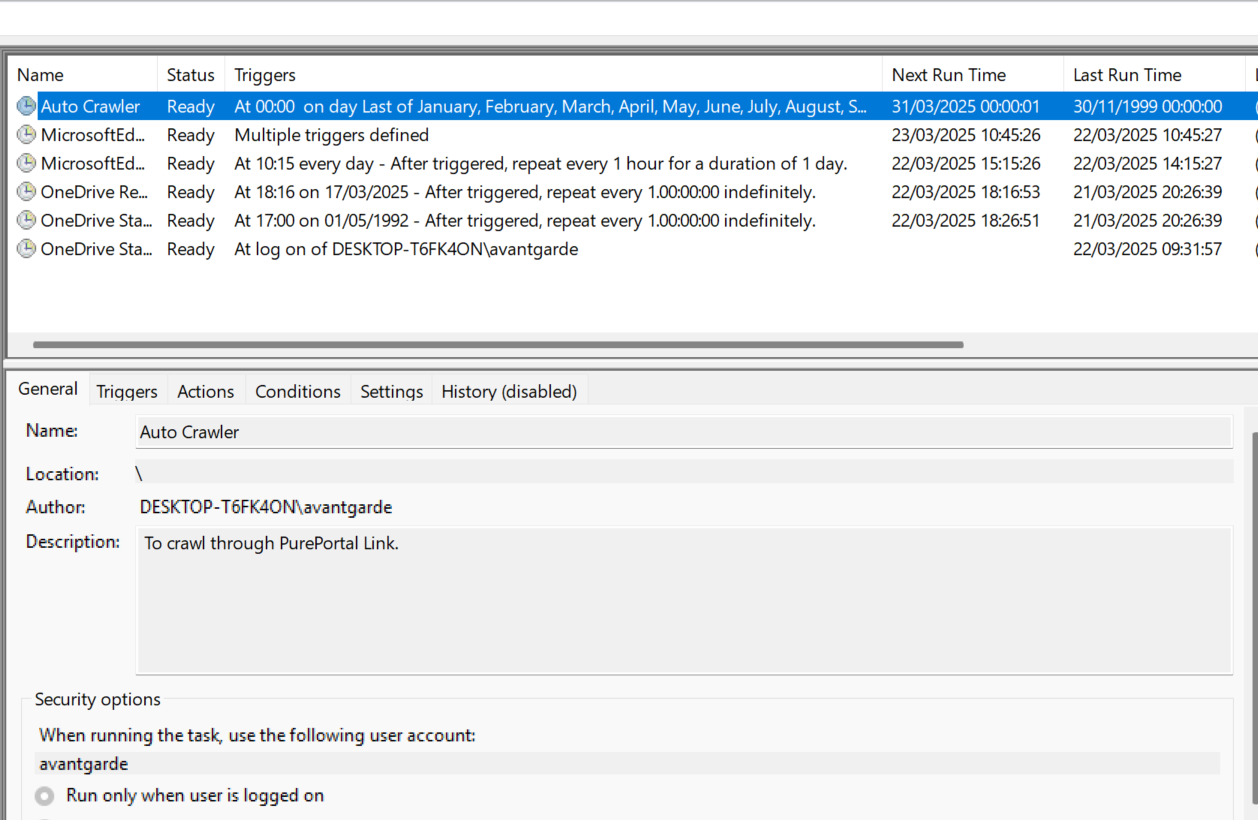


Figure 10 Final Search Engine

## 5. Scheduling with Task Scheduler

To automate crawling, we use Windows Task Scheduler:

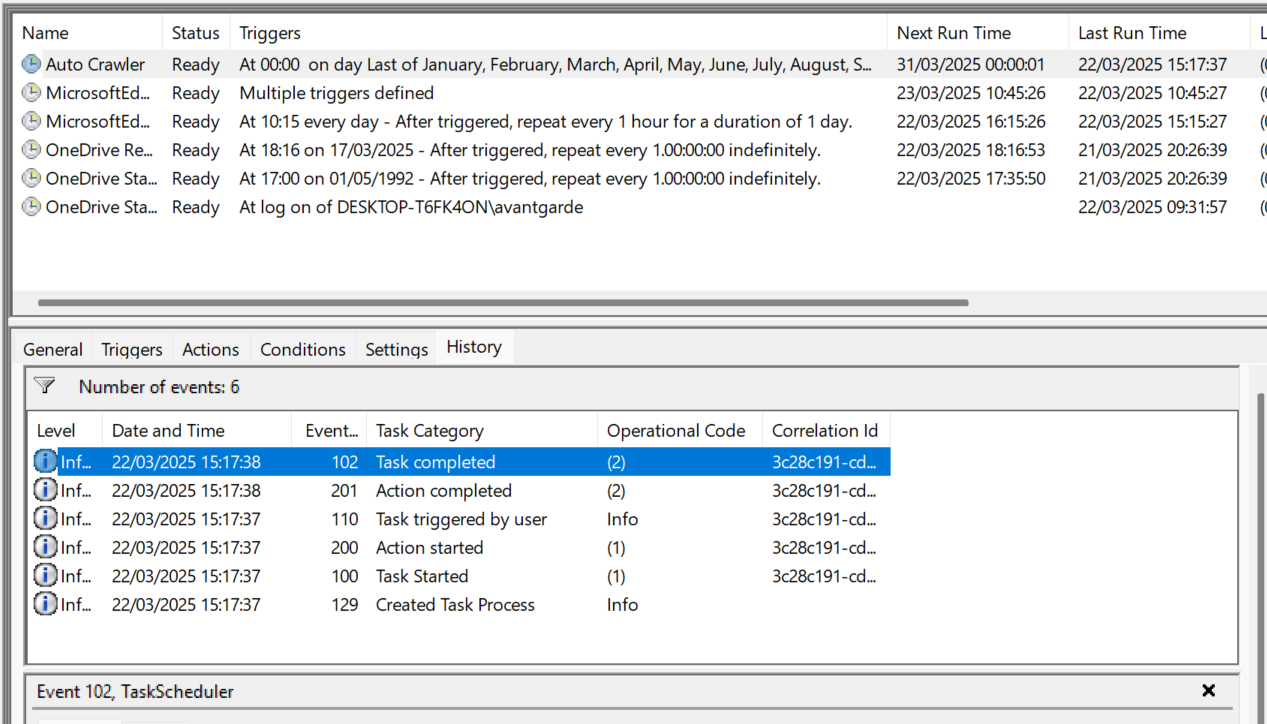


Figure 11 Scheduling with Task Scheduler

A run.bat file is created that executes craler.py and set it up in Task Scheduler to run once a month, ensuring that any new publication is uploaded if available. Additionally, there is a built-in history section that logs the details of each iteration.

## 6. Conclusion

This project provides a vertical search engine tailored for Coventry University's School of Economics, Finance, and Accounting. The system efficiently crawls, stores, and retrieves publication data, offering an intuitive search interface for users, with clickable links that direct users to the webpage directly.

# **Task 2: Subject Classification**

1. **Introduction**

Document classification is a crucial step in managing modern data. Document classification assigns a document to one or more pre-define categories based on its content. This helps to fasten the process of organizing and retrieving documents, by categorizing documents into specific classes it becomes easier to locate relevant information quickly, improving overall access and usability. This process assigns predefined classes to words, phrases and documents (Paaß, 2012). For example, in document classification, a document can be classified under the health field or the beauty field, depending on its content (El Barbary, 2020).

This project aims to develop document classification system that categorizes news article into Politics, Business, or Health. The classification is performed using a standard machine learning model, that classify new user-provided documents inro one of these categories accurately. The dataset contains 100 texts per category, collected from various news portal and stored in an excel file. The main goal of this project is to automatically classify texts into predefined categories, ensuring better organization and efficient retrieval of information.

1. **Data Collection**

The data for the model was collected from the publicly available news sources that includes categories such as Business, Politics and Health. For each of this category 100 data were collected. The data was gathered by manually copying and pasting the text in an excel file while maintaining ethical guidelines.

1. **Preprocessing**

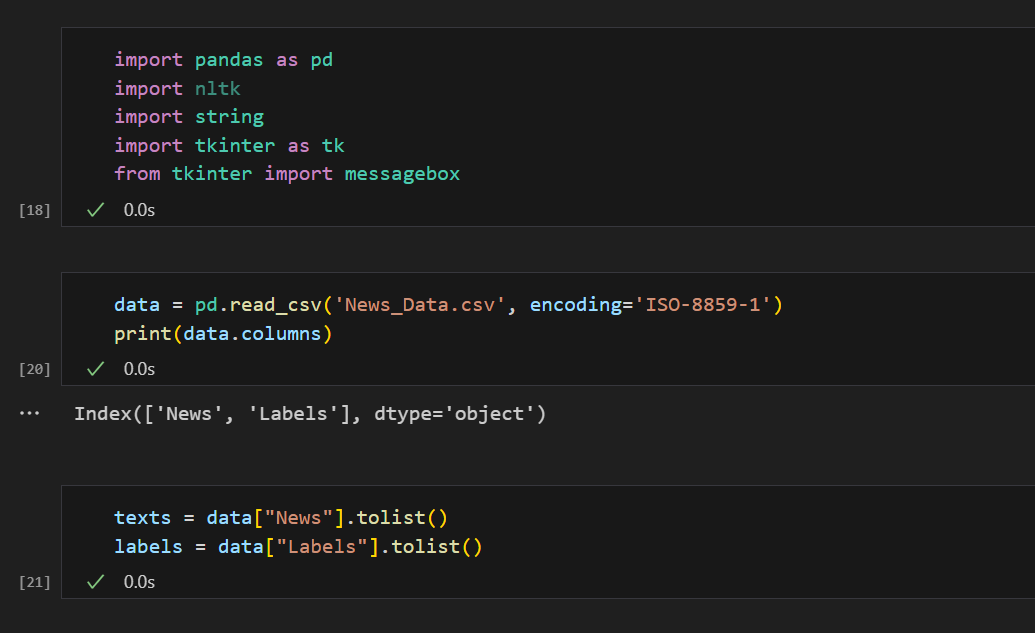


Figure 12: Importing Data Set

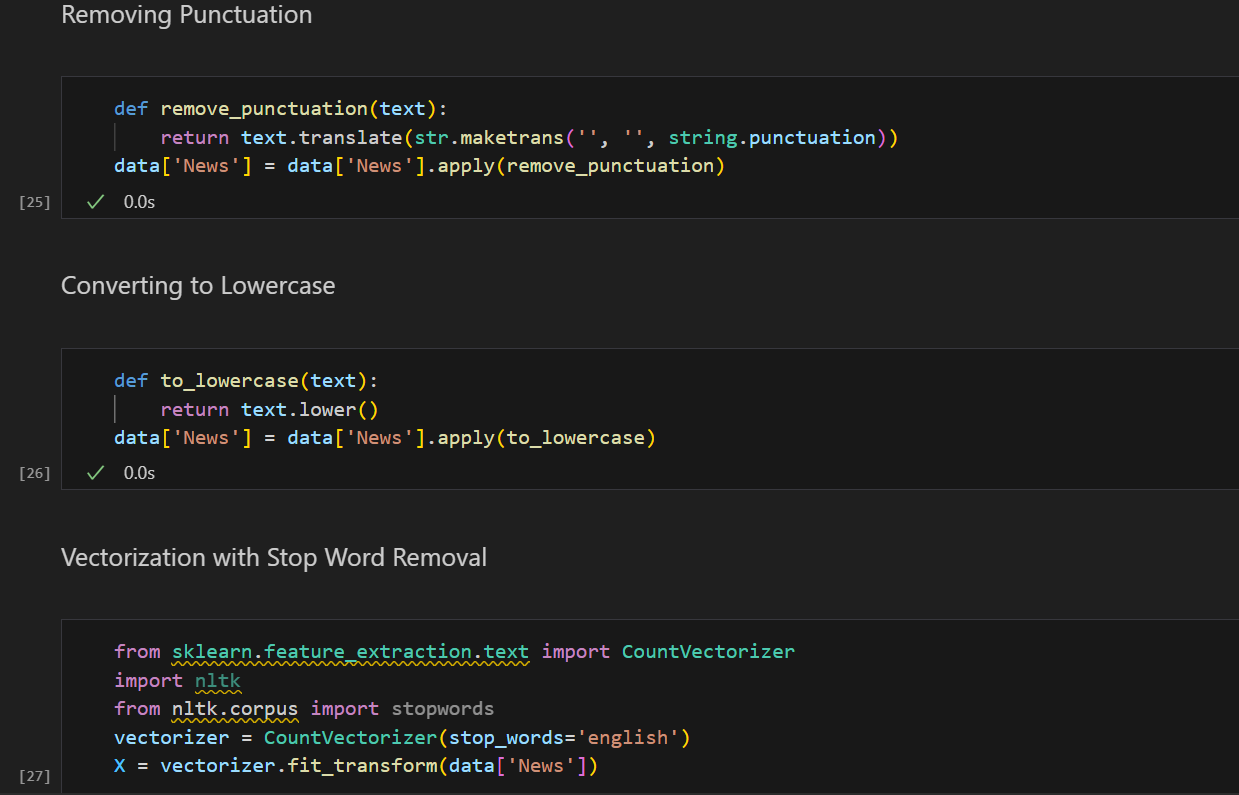


Figure 13 Preprocessing the Data Set

Prior to executing the classification tasks, several text preprocessing steps were undertaken. The first step in preprocessing of the text data is to remove punctuation marks like commas, periods, question marks, and exclamation points. These symbols do not add meaning to the text and can create inconsistencies during analysis. By removing them, we ensure that the words are processed in their purest form. This is applied to all the text in the 'News' column of the dataset. Next, step is converting text into lowercase. This helps maintain consistency by treating words like “The” and “the” as the same word. Without this step, uppercase and lowercase variations would be counted separately, leading to unnecessary complexity. This transformation makes the data cleaner and easier to analyze. Once the text is cleaned, it is converted into a numerical format so that a machine can understand it. This is done using a technique called vectorization, which transforms words into numbers based on how often they appear in the text. During this process, common words like "the," "is," and "a" are removed. These words, known as stop words, do not carry significant meaning and can dilute important patterns in the data. By filtering them out, we ensure that only the most relevant words are used for further analysis.

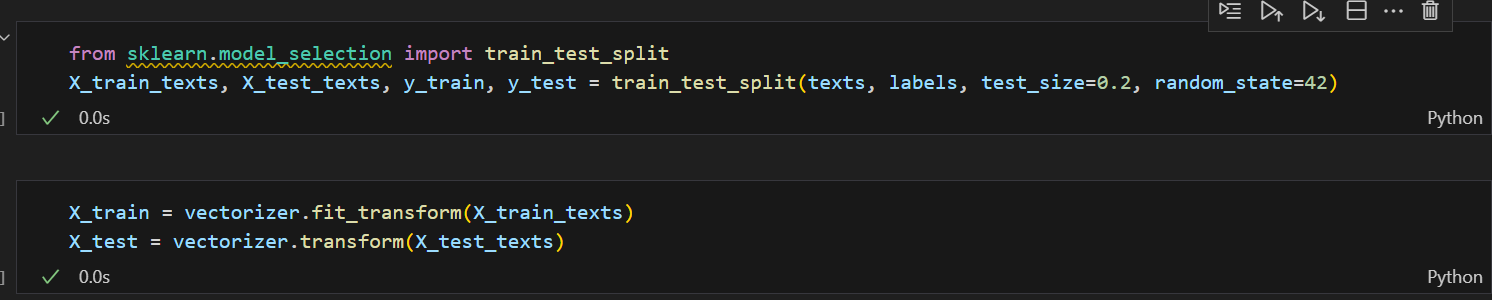


Figure 14: Splitting Data Set

Further, the data set is divided into training and testing, where 80% of the data is used as training data set and remaining 20% is used for testing. This ensures that the model learns from one set of data while being evaluated on another set it hasn't seen before. This followed by converting words into numbers for machine to understand. This is done by analyzing the training data to learn the unique words and their frequencies. Finally, the machine is trained on the numerical training data and later tested to see how well it performs on unseen text.

## Model Selection

### Logistic Regression

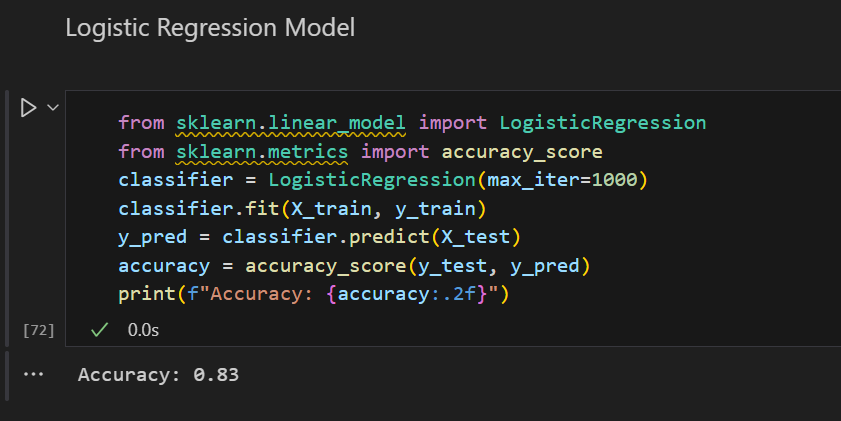


Figure 15: Logistic Regression

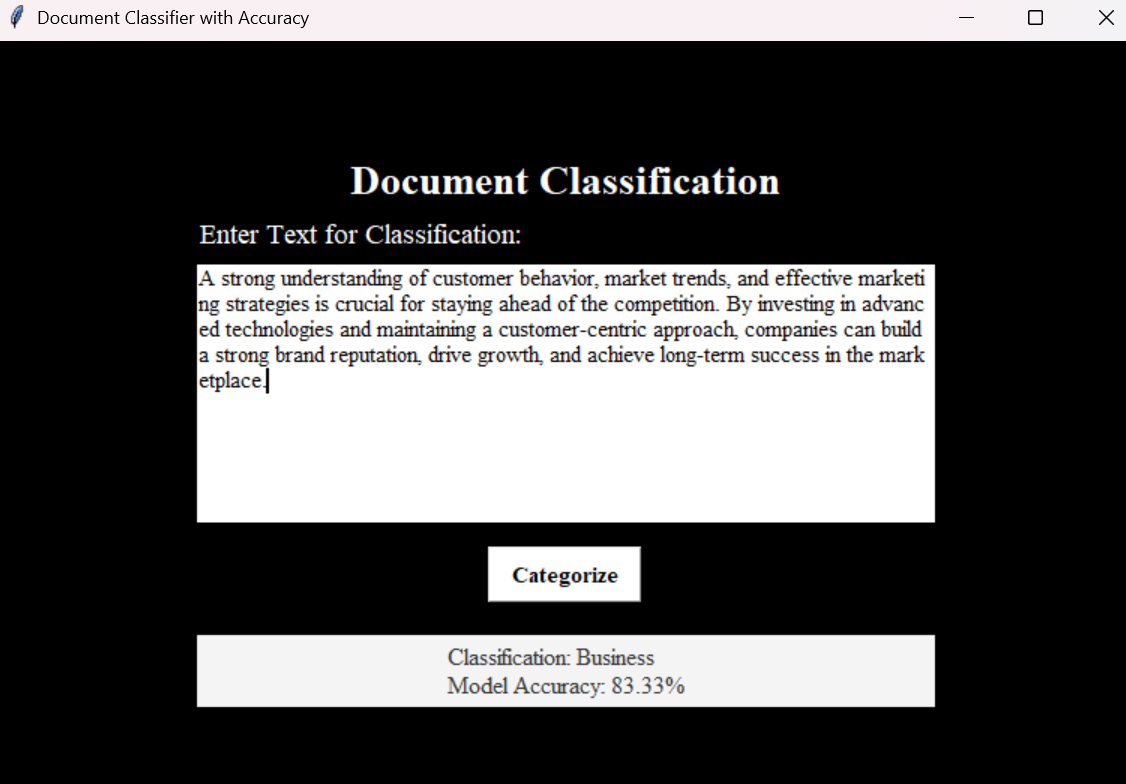


Figure 16 Document Classification Logistic Regression

A Logistic Regression model to classify documents is created. The model's performance is measured by accuracy, which tells us how often the model's predictions are correct. In this case, the model achieved an accuracy of 83%, meaning it correctly categorized the documents 83% of the time. A simple user interface is provided to interact with the model. We can input text, click a "Categorize" button, and see the model's classification result displayed on the screen. For example, if a user types any description it classifies into Business, Politics or Health as labeled in the data set. The text given to the model classifies it as "Business." Along with the classification result, the model's accuracy is shown, to showcase how reliable the model's predictions are.

### Navie Bayes Model

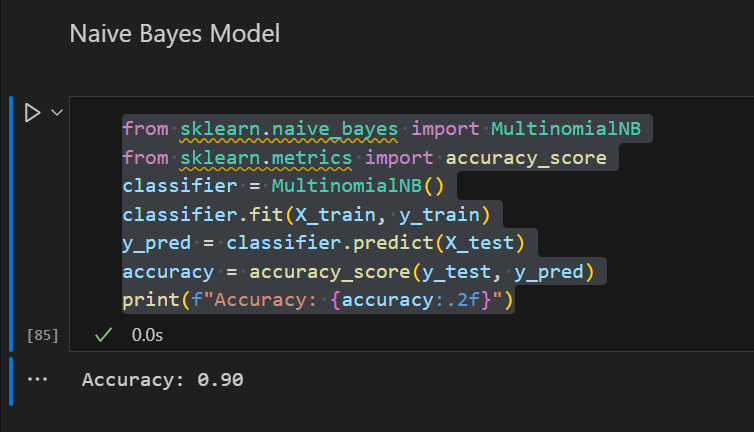


Figure 17 Naive Bayes Model

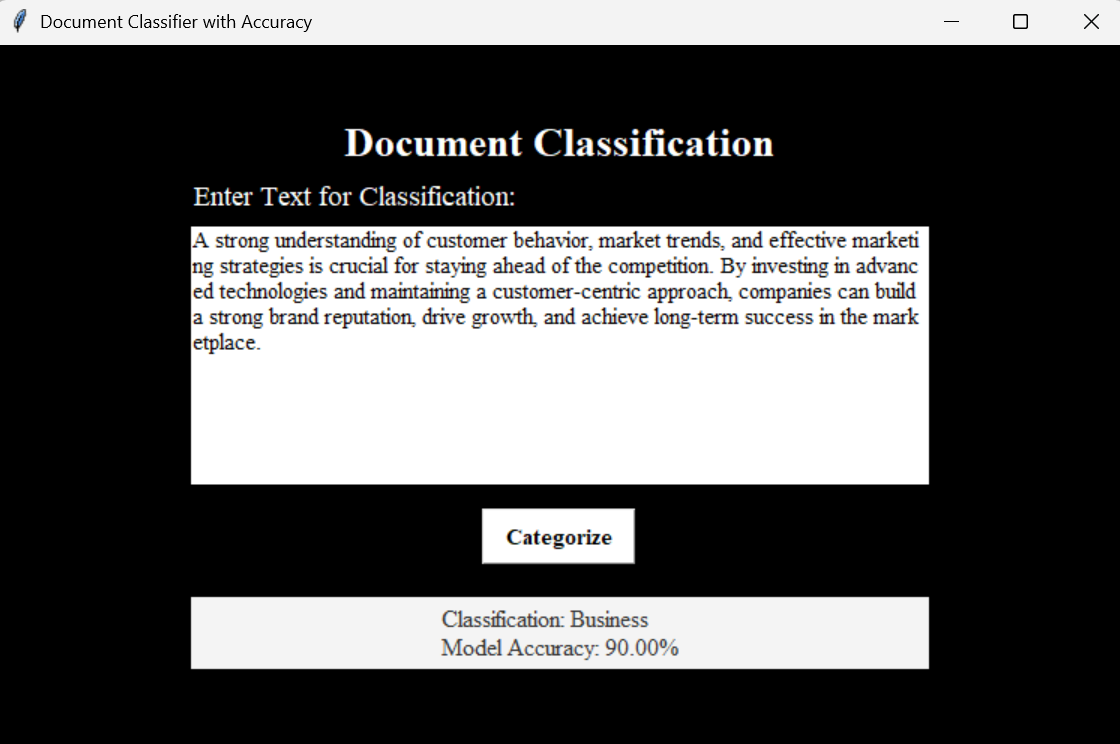


Figure 18 Document Classification Naive Bayes

A Multinomial Naive Bayes model to classify documents is created. The model's performance is measured by accuracy, which tells us how often the model's predictions are correct. In this case, the model achieved an accuracy of 90%, meaning it correctly categorized the documents 90% of the time. A simple user interface is provided to interact with the model. We can input text, click a "Categorize" button, and see the model's classification result displayed on the screen. For example, if a user types any description, the model classifies it into categories like Business, Politics, or Health, as labeled in the data set. The text given to the model classifies it as "Business." Along with the classification result, the model's accuracy is shown, to showcase how reliable the model's predictions are.

## Document Classification System

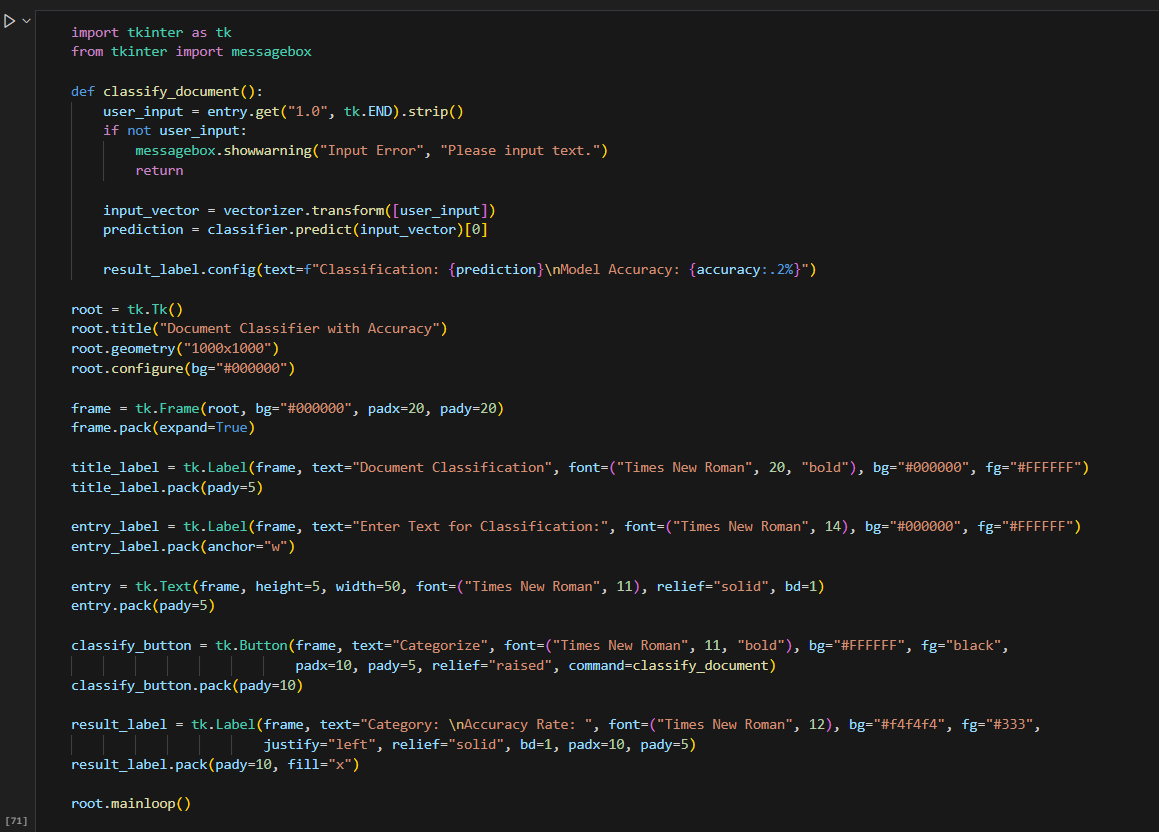
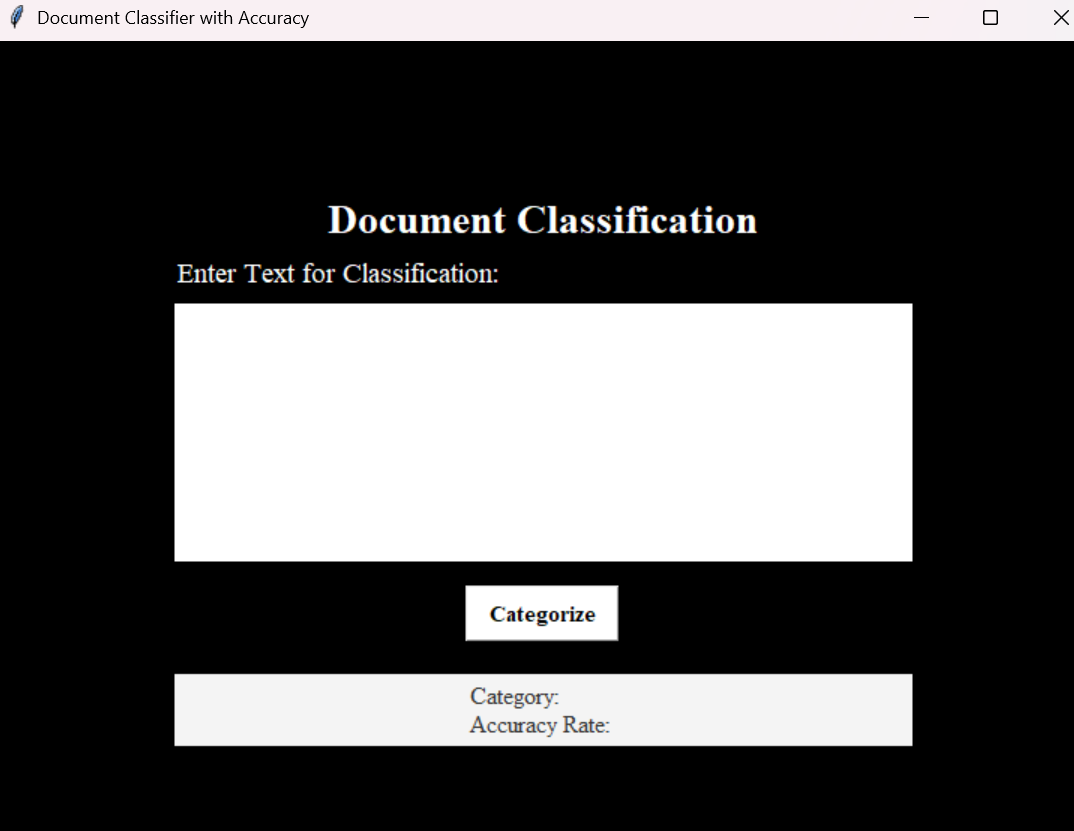


Figure 19 Document Classification System



This code creates a graphical user interface (GUI) application that allows users to input text, classify it using a pre-trained machine learning model, and display the classification result along with the model's accuracy. The application uses the Tkinter library to build the interface. When the user enters text and clicks the "Categorize" button, the application retrieves the input, checks if any text has been entered.

## Conclusion

This document classification system effectively categorizes texts into Politics, Business, or Health with high accuracy. Future improvements include expanding the dataset, integrating deep learning models, and enhancing feature extraction techniques. In this we used Naive Bayes and Logistic Regression to classify the text. The high accuracy was showcased in Naive Bayes as compared to the Logistic Regression.

# References

Paaß, G., 2012. Document classification, information retrieval, text and web mining. *Handbook of Technical Communication*, *8*, p.141.

El Barbary OG. Document classification in information retrieval system based on neutrosophic sets. Infinite Study; 2020.

Shettar, R. and Bhuptani, R., 2007. A vertical search engine–based on domain classifier. International Journal of Computer Science and Security, 2(4), pp.18-27.

GitHub Link: <https://github.com/shakyashreeya/Information_Retrieval_Assignment-.git>