

# Categorize Wikipedia pages

May 23, 2024

## Abstract

This project aims to gather information from Wikipedia pages and analyze its content to categorize them. Specifically, in this project we'll use web scraping to extract data from these pages. Then, we'll employ techniques to categorize them based on their content.

## 1 Introduction

Our project focuses on extracting and analyzing data from Wikipedia pages using web scraping techniques. Moreover, we want to categorize our pages based on their content. In order to explain our purpose, we are going to use an example. Let's take a look at [this page](#) in Wikipedia which is about Helium. As you can see, it is an article about the chemical element Helium. It talks about the history and the origin of this element. Moreover, it explains its characteristics and its applications. What we want as our objective is to explore all these text data and categorize this page in chemistry, periodic table, and element categories.

## 2 Objectives

Our main objectives are to:

- Gather raw data from Wikipedia pages as HTML.
- Use web scraping techniques to extract relevant information and remove unused data.
- Apply information retrieval techniques to identify page category based on its content.

## 3 Methodology

In this section we are going to talk about tools and techniques that we need for this project. We'll begin by fetching Wikipedia pages for analysis. Then, we'll employ web scraping methods using Python libraries to extract data from these pages. Finally, we'll use information retrieval techniques to determine the appropriate categories for the extracted content.

### 3.1 Gathering raw data

Since our input is a web-page address (aka URL), we need to fetch that page data by calling its address. Now, we don't have to manually copy and paste data from websites but a scraper can perform that task for us in a couple of seconds. In [Figure 1](#), you can see the process of fetching data from web pages.

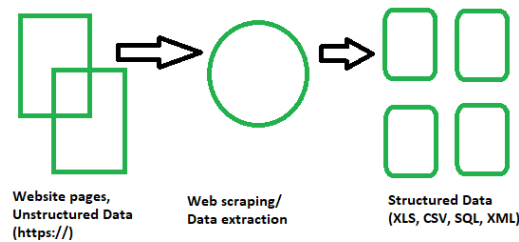


Figure 1: Gathering data from webpages

As our first step, we are going to use Python **requests** library to get a web page data as HTML format. The Python **requests** library is a powerful library for making HTTP requests in Python. It simplifies the process of sending HTTP requests and receiving responses, making it easy to interact with web services and APIs (application programming interface).

Here's a simple example of how to make a GET request using the **requests** library:

```

1 import requests
2
3 # fetching data from a Wikipedia page
4 response = requests.get("https://en.wikipedia.org/wiki/Helium")
5
6 # using response object to get the fetched details
7 print(response.status_code)
8 print(response.text)
9 print(response.content)

```

Listing 1: Example of getting a page content

In the code above, we opened a link and extracted its content as a string. Now I want you to use this example in order to get an input link and extract its content into a string variable.

**Warning:** Remember that if the input link is incorrent or returns an error, your Python code may crash. Therefore, make sure to handle these types of errors by using page status code field and Python try/exept feature.

## 3.2 Scraping Page Content

Python has several libraries to scrape web pages. Let's see the web scraping libraries in Python:

- Requests (HTTP for Humans) Library for Web Scraping: It is used for making various types of HTTP requests like GET, POST, etc. It is the most basic yet the most essential of all libraries.
- lxml Library for Web Scraping: lxml library provides super-fast and high-performance parsing of HTML and XML content from websites. If you are planning to scrape large datasets, this is the one you should go for.
- BeautifulSoup Library for Web Scraping: Its work involves creating a parse tree for parsing content. A perfect starting library for beginners and very easy to work with.

- Selenium Library for Web Scraping: Originally made for automated testing of web applications, this library overcomes the issue all the above libraries face i.e. scraping content from dynamically populated websites. This makes it slower and not suitable for industry-level projects.
- Scrapy for Web Scraping: The BOSS of all libraries, an entire web scraping framework which is asynchronous in its usage. This makes it blazing fast and increases efficiency.

Now we are going to use Python beautiful soup library in order to parse HTML content in to a list of paragraphs.

```

1 # import required modules
2 from bs4 import BeautifulSoup
3
4 # scrape webpage
5 soup = BeautifulSoup(response.content, 'html.parser')
6
7 # find all occurrence of p in HTML
8 # includes HTML tags
9 print(soup.find_all('p'))

```

Listing 2: Example of using BS4 library

The example above gets all 'p' tags that are paragraphs in a HTML page. Now I want you to migrate this example with your previous code in order to extract all paragraphs' contents from a Wikipedia page and save them into a text file.

### 3.3 Creating Index Table

In this section we are going to analyze our contents in order to categorize our Wikipedia pages. In order to do this, we are using a concept in Information Retrieval called Inverted Index.

An Inverted Index is a data structure used in information retrieval systems to efficiently retrieve documents or web pages containing a specific term or set of terms. In an inverted index, the index is organized by terms (words), and each term points to a list of documents or web pages that contain that term.

Generally there are three steps in building an inverted index:

1. Fetch the Document: Removing of Stop Words: Stop words are the most occurring and useless words in documents like "I", "the", "we", "is", and "an".
2. Stemming of Root Word: Whenever I want to search for "cat", I want to see a document that has information about it. But the word present in the document is called "cats" or "catty" instead of "cat". To relate both words, I'll chop some part of every word I read so that I could get the "root word". There are standard tools for performing this like "Porter's Stemmer".
3. Record Document IDs: If the word is already present add a reference of the document to index else creates a new entry. Add additional information like the frequency of the word, location of the word, etc.

In Figure 2 you can see the process of creating an inverted index from one or more documents.

Now I want you to read the text files from previous section, and create an inverted index for each of them. You don't need to merge all files; just create an inverted index for each Wikipedia page content. Finally, with an inverted index you can categorize your documents by analyzing

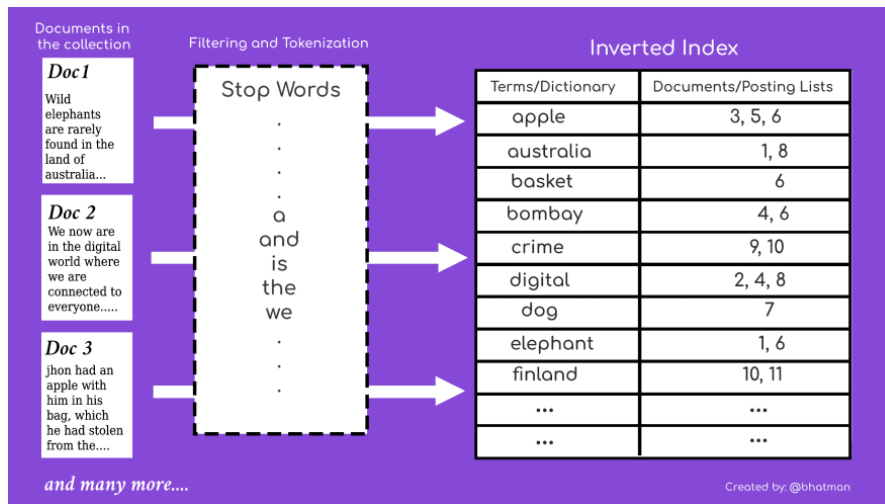


Figure 2: Inverted Index

the most words in their text. Make sure to suggest at least three categories for each Wikipedia page.

**Note:** There are some most used english words like 'is', 'the', etc. which are used alot in a paragraph. These words cannot be returned as a page categories since they are not an entity. Search through the internet to find how you can fix this issue.

## 4 Expected Outcomes

We anticipate achieving the following outcomes:

- Successful extraction of data from Wikipedia pages using **requests** module.
- Accurate determination of page titles based on content analysis using **beautiful soup**.
- Contribution to the understanding of web scraping and information retrieval inverted index technique.

## 5 References

- [geeksforgeeks.org/inverted-index](https://www.geeksforgeeks.org/inverted-index/).
- [geeksforgeeks.org/web-scraping](https://www.geeksforgeeks.org/web-scraping/)