
UITS

UNIVERSITY OF INFORMATION TECHNOLOGY AND SCIENCES

# Department of Computer Science and Engineering

Data Mining and Warehouse Lab

CSE 426

**Submission Date**
07/01/2025

## Report

**Project: Building a Simulation of a Search Engine**

**Submitted to**

Mrinmoy Biswas Akash
Lecturer
CSE Dept. UITS

**Submitted By**

Ishraq Uddin Chowdhury
2114951040
Section 8A2

**Project Title:** Building a Simulation of a Search Engine

## Introduction

Search engines are tools designed to index, search, and rank web pages or datasets based on queries. The project focuses on simulating a simple search engine using Python, leveraging datasets content.csv and graph.csv. This simulation involves building a graph, indexing content, and implementing query-based search capabilities.

## Basic Principles of a Search Engine

A search engine is a system designed to index, retrieve, and rank documents or web pages based on a user's query. The key principles behind a search engine are:

### 1. Crawling

Crawling involves systematically visiting and collecting data from web pages or documents. A crawler or "spider" navigates through links or datasets to gather content for analysis.

### 2. Indexing

Indexing organizes the crawled data into a structured format that allows for efficient searching. This typically involves creating:

- **Tokenized Words:** Extracting meaningful terms from text.
- **Inverted Index:** A data structure that maps each term to the documents it appears in.

### 3. Searching

The search engine matches the user's query with the indexed data to retrieve relevant results. Common search methods include:

- **Single-word Queries:** Returning documents containing a single query term.
- **Bag of Words:** Returning documents containing any of the terms in a multi-word query.

### 4. Ranking

The results are ranked based on their relevance to the query. Popular algorithms include:

- **PageRank:** Scores web pages based on their importance within the network of links.
- **TF-IDF:** Scores documents based on term frequency and inverse document frequency.

**Algorithms and Tools:**

- **Graph Representation:** Using networkx to model connections between web pages.
- **Tokenization and Stop Word Removal:** Extracting meaningful words and removing insignificant ones.
- **Inverted Index:** Mapping terms to the documents in which they appear.
- **Ranking:** Applying the PageRank algorithm to prioritize results.

**Methodology Behind Building the Search Engine**

**Step 1: Loading and Preprocessing Data**

The datasets (content.csv and graph.csv) are loaded into Python for analysis. The content.csv contains page content, while graph.csv represents the links between pages. Preprocessing includes cleaning text data and removing stop words to ensure meaningful terms are indexed.

**Step 2: Building the Graph**

The graph represents pages as nodes and hyperlinks as directed edges. Using the networkx library, the graph is constructed to analyze relationships between pages and compute PageRank scores.

**Step 3: Tokenization and Stop Word Removal**

To prepare content for indexing, the text is split into individual words (tokens). Stop words (e.g., "a," "the," "in") are removed using the nltk library to focus on informative terms.

**Step 4: Creating an Inverted Index**

An inverted index is constructed to map each unique word (token) to the document IDs where it appears. This structure enables fast retrieval of relevant documents for a given query.

**Step 5: Implementing Query-Based Search**

The search system is implemented in two modes:

1. **Single Word Search:** Returns documents containing the specific query word.
2. **Bag of Words Search:** Handles multi-word queries, retrieving documents containing any of the query terms.

**Step 6: Ranking Results**

The PageRank algorithm is applied to rank documents. Pages with higher importance, as determined by their network connectivity, are ranked higher in the results.

## Implementation

Load and Process Data and mount Datasets to Google drive

We first import the libraries and read the datasets **content.csv** and **graph.csv**.

**Code:**

```
!pip install nltk
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (3.9.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk) (4.67.1)
```

```
[55] import nltk
     nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
True
```

```
[86] # Import required libraries
     import networkx as nx
     import pandas as pd
     import matplotlib.pyplot as plt
     from nltk.tokenize import word_tokenize
     from nltk.corpus import stopwords
     import string
     from google.colab import drive
     import nltk
     import re
```

```
[2]  # Mount Google Drive
     drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[3]
     # Download NLTK resources
     nltk.download('punkt')
     nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

```
[6]  # Load datasets
     graph_file = "/content/drive/MyDrive/Data Mining Final Project/graph.csv"
     content_file = "/content/drive/MyDrive/Data Mining Final Project/content.csv"
```

```
[7]  edges_df = pd.read_csv(graph_file)
     content_df = pd.read_csv(content_file)
```

**Task 01: Create a graph from graph.csv and load contents from content.csv**

We used the networkx library to visualize the connections between nodes.

**Code:**

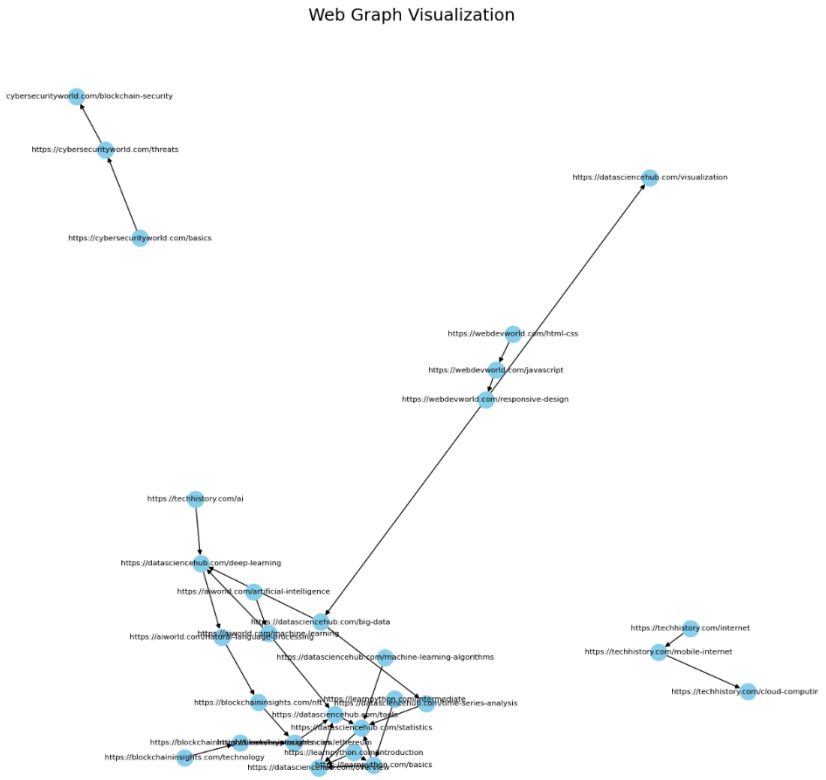
```
[8]  web_graph = nx.DiGraph()
     for _, row in edges_df.iterrows():
         web_graph.add_edge(row["Source"], row["Target"])

[9]  # Load content into a dictionary
     web_content = dict(zip(content_df["URL"], content_df["Content"]))

[14] # Display graph info manually
     print(f"Number of nodes: {web_graph.number_of_nodes()}")
     print(f"Number of edges: {web_graph.number_of_edges()}")
     print(f"Nodes: {list(web_graph.nodes())}")
     print(f"Edges: {list(web_graph.edges())}")
```

**Task 02: Visualize the graph**

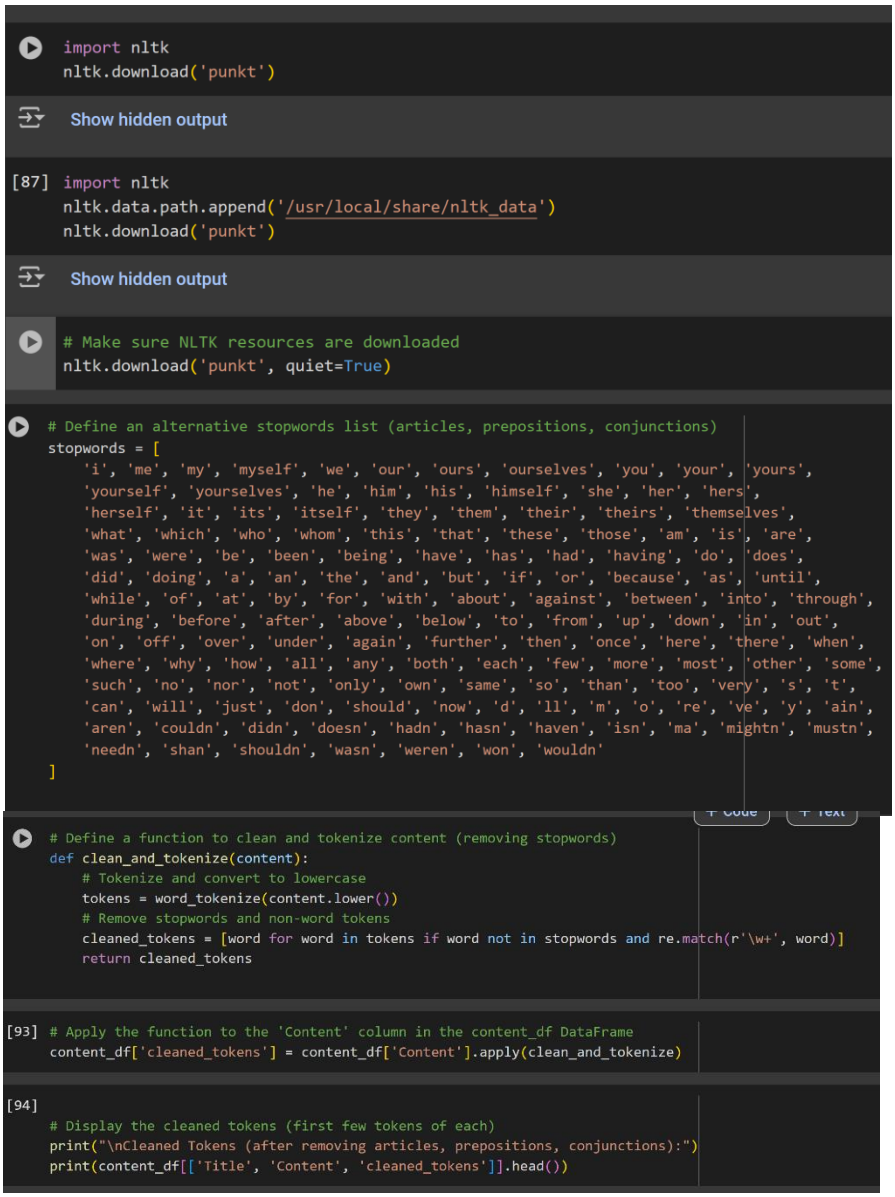
**Code:**



## Task 03 (Bonus): Tokenize and Remove Stop Words

Tokenization splits text into words, and stop word removal eliminates non-informative words.

## Code:



```
import nltk
nltk.download('punkt')
```

```
Show hidden output
```

```
[87] import nltk
nltk.data.path.append('/usr/local/share/nltk_data')
nltk.download('punkt')
```

```
Show hidden output
```

```
# Make sure NLTK resources are downloaded
nltk.download('punkt', quiet=True)
```

```
# Define an alternative stopwords list (articles, prepositions, conjunctions)
stopwords = [
    'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your', 'yours',
    'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', 'her', 'hers',
    'herself', 'it', 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves',
    'what', 'which', 'who', 'whom', 'this', 'that', 'these', 'those', 'am', 'is', 'are',
    'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does',
    'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until',
    'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through',
    'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out',
    'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when',
    'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some',
    'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't',
    'can', 'will', 'just', 'don', 'should', 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain',
    'aren', 'couldn', 'didn', 'doesn', 'hadn', 'hasn', 'haven', 'isn', 'ma', 'mightn', 'mustn',
    'needn', 'shan', 'shouldn', 'wasn', 'weren', 'won', 'wouldn'
]
```

```
# Define a function to clean and tokenize content (removing stopwords)
def clean_and_tokenize(content):
    # Tokenize and convert to lowercase
    tokens = word_tokenize(content.lower())
    # Remove stopwords and non-word tokens
    cleaned_tokens = [word for word in tokens if word not in stopwords and re.match(r'\w+', word)]
    return cleaned_tokens
```

```
[93] # Apply the function to the 'Content' column in the content_df DataFrame
content_df['cleaned_tokens'] = content_df['Content'].apply(clean_and_tokenize)
```

```
[94]
# Display the cleaned tokens (first few tokens of each)
print("\nCleaned Tokens (after removing articles, prepositions, conjunctions):")
print(content_df[['Title', 'Content', 'cleaned_tokens']].head())
```

**Task 04: Build an Inverted Index**
An inverted index maps each term to the documents in which it appears.

**Code**:

```python
# prompt: Using dataframe content_df: Build an inverted index

import pandas as pd

def build_inverted_index(df):

    inverted_index = {}
    # Iterate through each document in the DataFrame
    for index, row in df.iterrows():
        url = row['URL']
        content = row['Content']

        # Tokenize the content (split into words)
        words = content.lower().split()

        # Update the inverted index for each word
        for word in words:
            if word not in inverted_index:
                inverted_index[word] = []
            inverted_index[word].append(url)
    return inverted_index

# Build the inverted index from the content_df DataFrame.
inverted_index = build_inverted_index(content_df)

# Print the inverted index (optional)
for word, urls in inverted_index.items():
    print(f"{word}: {urls}")
```

**Task 05: Single Word Query Search with PageRank**
We implemented a single-word search system that uses the PageRank algorithm for ranking results.

**Code:**

```
[95] pagerank_scores = nx.pagerank(web_graph, alpha=0.85)

     def single_word_query(word, index, scores):
         word = word.lower()
         if word in index:
             urls = index[word]
             ranked_urls = sorted(urls, key=lambda url: scores.get(url, 0), reverse=True)
             return ranked_urls
         return []
```

```
[97] query_word = input("\nEnter a single-word query: ")
     results = single_word_query(query_word, inverted_index, pagerank_scores)
     print(f"\nSearch Results for '{query_word}':")
     for url in results:
         print(f"- {url} (PageRank: {pagerank_scores[url]:.4f})")
```

```
Enter a single-word query: python

Search Results for 'python':
- https://datasciencehub.com/tools (PageRank: 0.2203)
- https://learnpython.com/basics (PageRank: 0.0156)
- https://learnpython.com/basics (PageRank: 0.0156)
- https://learnpython.com/introduction (PageRank: 0.0069)
- https://learnpython.com/introduction (PageRank: 0.0069)
```

Output: A ranked list of pages matching the query based on their PageRank scores.

**Task 06 (Bonus): Bag of Words Query Search**

This approach returns pages containing any of the query terms.

```
[98] def bag_of_words_query(words, index, scores):
         words = [word.lower() for word in words]
         relevant_urls = set()
         for word in words:
             if word in index:
                 relevant_urls.update(index[word])
         ranked_urls = sorted(relevant_urls, key=lambda url: scores.get(url, 0), reverse=True)
         return ranked_urls
```

```
query_words = input("\nEnter a multi-word query (space-separated): ").split()
results = bag_of_words_query(query_words, inverted_index, pagerank_scores)
print(f"\nSearch Results for '{' '.join(query_words)}':")
for url in results:
    print(f"- {url} (PageRank: {pagerank_scores[url]:.4f})")
```

```
Enter a multi-word query (space-separated): python is

Search Results for 'python is':
- https://datasciencehub.com/tools (PageRank: 0.2203)
- https://datasciencehub.com/overview (PageRank: 0.2014)
- https://learnpython.com/basics (PageRank: 0.0156)
- https://webdevworld.com/javascript (PageRank: 0.0127)
- https://aiworld.com/machine-learning (PageRank: 0.0127)
- https://blockchaininsights.com/technology (PageRank: 0.0069)
- https://webdevworld.com/html-css (PageRank: 0.0069)
```

**Output:** A ranked list of pages matching any of the query terms.

**Future Improvements:**

- Advanced ranking using TF-IDF.
- Support for phrase queries and semantic search.
- Scalability to handle larger datasets.

**Project Link Github:**

https://github.com/ishraqX/Data-Mining/

**Conclusion**

The project demonstrates the fundamental principles of search engines:

- Graph representation is used to understand relationships between pages.
- Efficient data structuring with inverted indices.
- Query-based search systems with ranking mechanisms.