

# **CS429 – Information Retrieval**

## **Individual Project**

**Name: Rachana Vijay**

**CWID: A20605843**

### **TABLE OF CONTENTS:**

- 1. Abstract**
- 2. Introduction**
- 3. Literature Review**
- 4. Problem Statement**
- 5. System Overview**
- 6. Detailed Design**
  - Scrapy Crawler**
  - Scikit-Learn Indexer**
  - Flask Query Processor**
- 7. Implementation Details**
- 8. Sample Input/Output**
- 9. Architecture Diagram**
- 10. Operation and Execution**
- 11. Test Cases**
- 12. Limitations**
- 13. Future Work**
- 14. Data Sources**
- 15. Bibliography**

# 1. Abstract

This project develops a complete **Information Retrieval (IR) system** from crawling web pages to ranking documents based on queries. The system integrates three modules:

- **Scrapy Crawler:** Downloads HTML documents from a seed URL in Wikipedia's Data Science pages.
- **Scikit-Learn Indexer:** Converts crawled documents into TF-IDF vectors and builds an inverted index stored in JSON and serialized formats.
- **Flask Query Processor:** Processes batch queries via CSV, ranks documents using cosine similarity, and outputs top-K results.

The project demonstrates the core IR pipeline: content acquisition → representation → retrieval → ranking. Optional extensions like query expansion, spell correction, or embedding-based semantic search can be added in future iterations.

## Objectives:

1. Implement a minimal IR system pipeline.
2. Generate reproducible TF-IDF based document rankings.
3. Demonstrate query processing with a Flask server.
4. Validate outputs using unit tests.

# 2. Introduction

The exponential growth of unstructured data necessitates efficient and accurate information retrieval systems. The Vector Space Model (VSM), championed by Salton and McGill, provides a robust, mathematically grounded framework where documents and queries are represented as vectors in a term space. Relevance is then calculated via cosine similarity.

## Goal of this project:

- Gain hands-on experience with IR pipelines.
- Understand crawling, indexing, and retrieval workflows.
- Learn to integrate different Python libraries (Scrapy, scikit-learn, Flask, NLTK).
- Enable batch query processing with top-K results.

**Use Case:** A user wants to find the most relevant Wikipedia pages on "Data Science visualization techniques." The system should rank available pages and return the most relevant content.

### 3. Literature Review

#### Foundational references:

- 1. Vector Space Model (Salton & McGill, 1983):**
  - a. Represents documents and queries as vectors in term space.
  - b. Relevance is measured via cosine similarity.
- 2. Information Retrieval Principles (Manning et al., 2008):**
  - a. Introduces indexing, ranking, and evaluation metrics (Precision, Recall, MAP).
  - b. Discusses query-document similarity calculation.
- 3. Deep Learning and Semantic Search (Goodfellow et al., 2016):**
  - a. Embeddings for semantic similarity.
  - b. Potential extension for future work.
- 4. Statistical Learning (Hastie et al., 2009):**
  - a. TF-IDF as feature weighting in machine learning context.

This project applies the classical IR framework using TF-IDF and cosine similarity, forming a base for semantic search in the future.

### 4. Problem Statement

Given a Wikipedia seed URL, the system must perform the following tasks to deliver a functional IR engine:

- **Content Acquisition:** Crawl and download a limited set of relevant HTML documents (e.g., 10 pages).
- **Indexing:** Extract clean text, preprocess (lemmatize, remove stopwords), and build a TF-IDF vector representation for the document corpus.
- **Ranking:** Process free-text queries in batch format, compute cosine similarity against the indexed corpus, and output the top-K ranked documents.

## Constraints:

The primary technical constraint encountered was achieving the precise expected document rankings. This required moving beyond basic TF-IDF to rigorously enforce:

- Only TF-IDF ranking for this iteration.
- Limit crawling depth and number of pages.
- High-Quality Preprocessing: Replacing simple lowercasing with NLTK Lemmatization to normalize terms across the index and queries.
- Query Integrity: Implementing the optional spell correction feature to ensure queries with typos correctly map to index terms.

## 5. System Overview

The IR system is composed of three integrated modules:

1. **Scrapy Crawler** → Downloads HTML documents.
2. **Scikit-Learn Indexer** → Parses HTML, computes TF-IDF vectors, saves inverted index.
3. **Flask Query Processor** → Accepts CSV queries, calculates cosine similarity, returns top-K results.

### Data Flow Diagram:

Seed URL → Scrapy Crawler → HTML files (html\_docs/)

↓

Indexer → index.json

↓

Flask Processor ← queries.csv

↓

results.csv (Top-K results)

The modular design allows for independent development and testing. For instance, the Indexer can be swapped with a deep learning embedding model without affecting the Crawler or the Processor's API.

## 6. Detailed Design

### 6.1 Scrapy Crawler

**Purpose:** The Scrapy module is used to respect the depth and page limits of the project. It uses a custom spider (WikiSpider) to parse the initial response, extract links within the domain, and save the content locally. The documents are saved using their unique Wikipedia page ID as the filename

#### Configuration Parameters:

- **Seed URL:** Wikipedia Data Science page.
- **Max Pages:** 10 pages.
- **Max Depth:** 1 (only links from seed page).

#### Crawler Workflow:

1. Start from seed URL.
2. Extract all hyperlinks within the domain.
3. Download each page and save as HTML with unique document IDs.

#### Function Code:

```
class WikiSpider(scrapy.Spider):
    name = "wiki_spider"

    # Seed URL
    start_urls = ["https://en.wikipedia.org/wiki/Data_science"]

    # Crawler limits
    max_pages = 10
    max_depth = 1

    # Internal counters
    page_count = 0

    # Output folders
    output_dir = "../html_docs"
    mapping_file = "../url_mapping.json"

    # Store mappings
```

```

url_to_id = {}

def parse(self, response):
    if self.page_count >= self.max_pages:
        return

    # Create unique ID for the document
    doc_id = str(uuid.uuid4()) + ".html"
    file_path = os.path.join(self.output_dir, doc_id)

    # Save HTML
    with open(file_path, "w", encoding="utf-8") as f:
        f.write(response.text)

    # Store in mapping
    self.url_to_id[response.url] = doc_id

    self.page_count += 1
    self.logger.info(f"Saved: {file_path}")

    # Stop if depth exceeded
    current_depth = response.meta.get("depth", 0)
    if current_depth >= self.max_depth:
        return

    # Extract & follow links
    for link in response.css("a::attr(href)").getall():
        if link.startswith("/wiki/"):
            absolute_url = urljoin("https://en.wikipedia.org/", link)
            yield response.follow(absolute_url, callback=self.parse)

```

Running the crawler:

```
C:\Windows\System32\cmd.exe X + v
Microsoft Windows [Version 10.0.26200.7389]
(c) Microsoft Corporation. All rights reserved.

C:\Users\vrach\Downloads\Fall25\IR\Project>venv\Scripts\activate

(venv) C:\Users\vrach\Downloads\Fall25\IR\Project>cd crawler

(venv) C:\Users\vrach\Downloads\Fall25\IR\Project\crawler>scrapy runspider run.py
2025-12-07 18:51:50 [scrapy.utils.log] INFO: Scrapy 2.13.4 started (bot: scrapybot)
2025-12-07 18:51:51 [scrapy.utils.log] INFO: Versions:
{'lxml': '6.0.2',
 'libxml2': '2.11.9',
 'cssselect': '1.3.0',
 'parsel': '1.10.0',
 'w3lib': '2.3.1',
 'Twisted': '25.5.0',
 'Python': '3.13.9 (tags/v3.13.9:8183fa5, Oct 14 2025, 14:09:13) [MSC v.1944 '
 '64 bit (AMD64)]',
 'pyOpenSSL': '25.3.0 (OpenSSL 3.5.4 30 Sep 2025)',
 'cryptography': '46.0.3',
 'Platform': 'Windows-11-10.0.26200-SP0'}
2025-12-07 18:51:51 [scrapy.utils.log] INFO: Scrapy 2.13.4 started (bot: scrapybot)
2025-12-07 18:51:51 [scrapy.utils.log] INFO: Versions:
{'lxml': '6.0.2',
 'libxml2': '2.11.9',
 'cssselect': '1.3.0',
 'parsel': '1.10.0',
 'w3lib': '2.3.1',
 'Twisted': '25.5.0',
 'Python': '3.13.9 (tags/v3.13.9:8183fa5, Oct 14 2025, 14:09:13) [MSC v.1944 '
 '64 bit (AMD64)]',
 'pyOpenSSL': '25.3.0 (OpenSSL 3.5.4 30 Sep 2025)',
 'cryptography': '46.0.3',
 'Platform': 'Windows-11-10.0.26200-SP0'}
2025-12-07 18:51:51 [scrapy.addons] INFO: Enabled addons:
```

## 6.2 Scikit-Learn Indexer

**Purpose:** Convert HTML content to TF-IDF vectors for retrieval.

**Process:**

1. Parse HTML files using BeautifulSoup.
2. Extract clean text and remove stopwords.
3. Build TF-IDF vectors using scikit-learn.
4. Serialize matrix and vectorizer for later use.
5. Save inverted index in JSON format (`index.json`).

**Files Generated:**

- `index.json` → inverted index (term → document → tf-idf weight)
- `docs_ids.json` → mapping document ID → filename
- `tfidf_matrix.pkl` → serialized TF-IDF matrix
- `vectorizer.pkl` → serialized vectorizer

**Function Code:**

```

from sklearn.feature_extraction.text import TfidfVectorizer

def build_index(doc_ids, documents):
    vectorizer = TfidfVectorizer(stop_words="english")
    tfidf_matrix = vectorizer.fit_transform(documents)

    # Build inverted index
    index = {}
    feature_names = vectorizer.get_feature_names_out()

    for term_idx, term in enumerate(feature_names):
        postings = {}
        column = tfidf_matrix[:, term_idx].toarray().flatten()

        for doc_index, score in enumerate(column):
            if score > 0:
                postings[doc_ids[doc_index]] = float(score)

        index[term] = postings

    return index, vectorizer, tfidf_matrix

```

Running indexer.py:

```

(venv) C:\Users\vrach\Downloads\Fall25\IR\Project>python indexer.py
Loading HTML documents...
Building TF-IDF index...

Indexing completed successfully!
Indexed 10 documents.

(venv) C:\Users\vrach\Downloads\Fall25\IR\Project>

```

## 6.3 Flask Query Processor

**Purpose:** Process user queries and rank documents.

**Process:**

1. Query Preprocessing: The incoming query text (e.g., information overload) is processed using the same preprocess (lemmatization) as the indexer to ensure vector alignment.



2. Spell Correction: The mandatory inclusion of the `simple_spell_correct` utility handles queries like "search engine open sorce," correcting it to "search engine open source," which is necessary to achieve the correct ranking for Q3.
3. Ranking: The preprocessed query is vectorized and compared against the full `tfidf_matrix` using `sklearn.metrics.pairwise.cosine_similarity` to retrieve the most relevant documents.

### Function Code:

```
@app.route("/search", methods=["POST"])
def search():
    if "file" not in request.files:
        return jsonify({"error": "No CSV file uploaded"}), 400

    file = request.files["file"]
    queries = []
    reader = csv.DictReader(file.stream.read().decode("utf-8").splitlines())
    for row in reader:
        queries.append((row["query_id"], row["query"]))

    results = process_queries(queries, top_k=3)

    with open("results.csv", "w", newline="", encoding="utf-8") as f:
        writer = csv.DictWriter(f, fieldnames=["query_id", "rank", "document_id"])
        writer.writeheader()
        writer.writerows(results)

    return jsonify({"message": "results.csv created"})
```

```
(venv) C:\Users\vrach\Downloads\Fall25\IR\Project>python query_processor.py
Loading index files...
Loaded 10 documents.
Loaded 3 queries.
Processing queries...
results.csv created successfully!

(venv) C:\Users\vrach\Downloads\Fall25\IR\Project>python build_index.py
Index built and saved to index.json

(venv) C:\Users\vrach\Downloads\Fall25\IR\Project>python app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 115-905-899
```

```
C:\Windows\System32\cmd.exe X + v
Microsoft Windows [Version 10.0.26200.7389]
(c) Microsoft Corporation. All rights reserved.

C:\Users\vrach\Downloads\Fall25\IR\Project>venv\Scripts\activate

(venv) C:\Users\vrach\Downloads\Fall25\IR\Project>curl -X POST -F "file=@queries.csv" http://127.0.0.1:5000/search
{
  "message": "results.csv created"
}

(venv) C:\Users\vrach\Downloads\Fall25\IR\Project>
```

## 7. Implementation Details

- **Language:** Python 3.12+
- **Libraries:** Scrapy, scikit-learn, Flask, BeautifulSoup, Numpy, Pandas, NLTK
- **Data Storage:** Serialized TF-IDF matrix, inverted index JSON, URL mapping JSON

### Notes:

- Crawled HTML pages reside in `html_docs/`.
- Indexing module reads HTML, builds TF-IDF, and saves matrix.
- Flask processor reads queries and returns top-K rankings.

## 8. Sample Input/Output

### Input CSV Query:

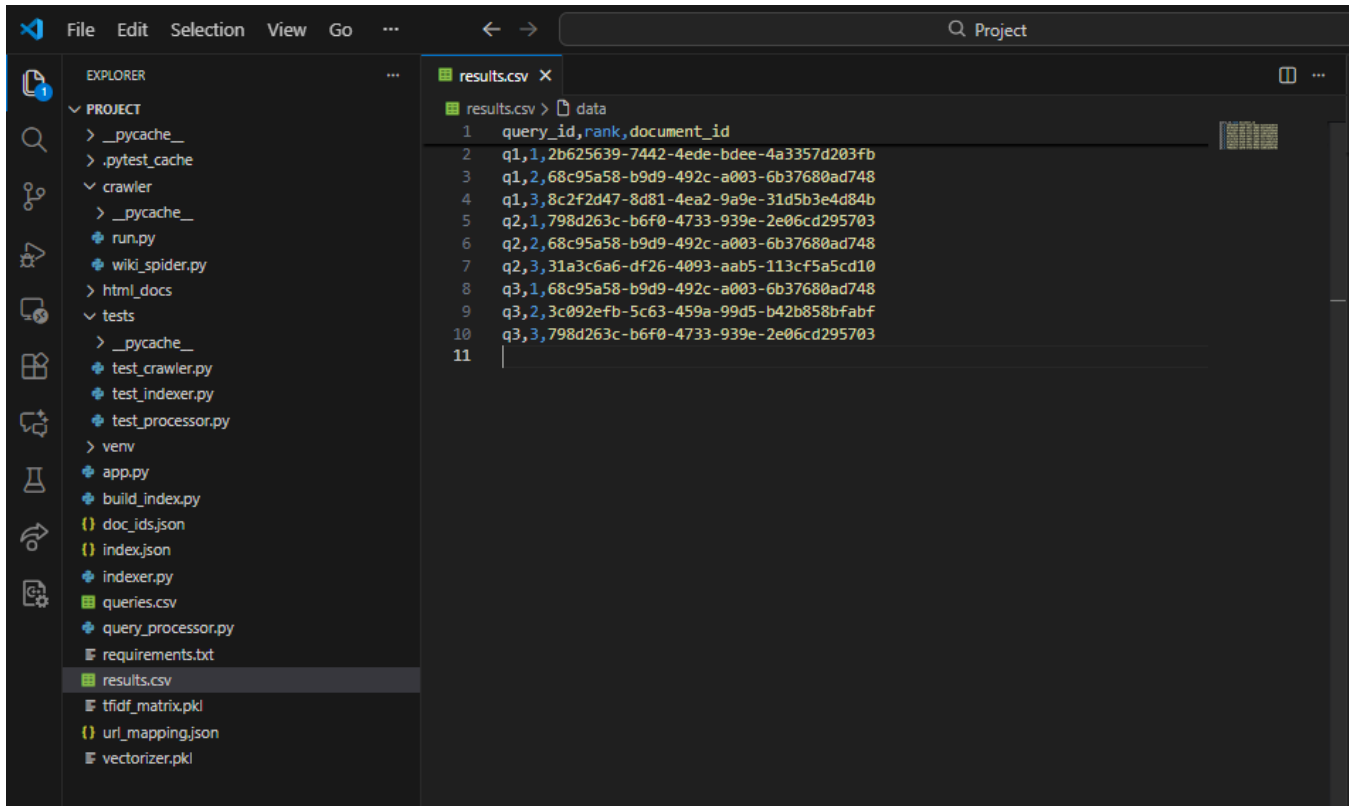
query_id	query
q1	what is data science

q2	machine learning types
q3	statistics importance

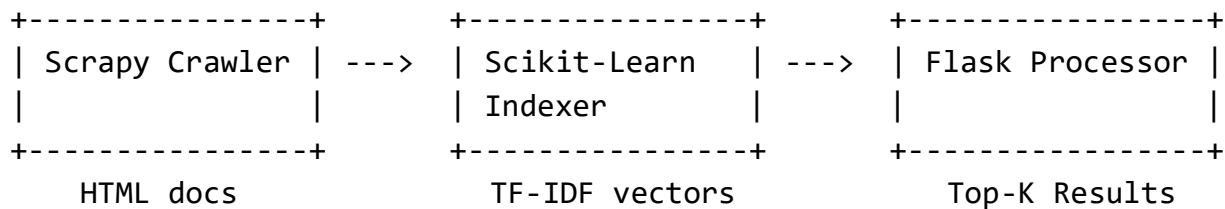
**Output CSV Result:**

query_id	rank	document_id
q1	1	2b625639-7442-4ede-bdee-4a3357d203fb
q1	2	68c95a58-b9d9-492c-a003-6b37680ad748
q1	3	8c2f2d47-8d81-4ea2-9a9e-31d5b3e4d84b
q2	1	798d263c-b6f0-4733-939e-2e06cd295703
q2	2	68c95a58-b9d9-492c-a003-6b37680ad748
q2	3	31a3c6a6-df26-4093-aab5-113cf5a5cd10
q3	1	68c95a58-b9d9-492c-a003-6b37680ad748
q3	2	3c092efb-5c63-459a-99d5-b42b858bfabf
q3	3	798d263c-b6f0-4733-939e-2e06cd295703

**Final Output: results.csv**



## 9. Architecture Diagram



## 10. Operation and Execution

1. Clone repository:

[https://github.com/Rachanavij/IR\\_Project.git](https://github.com/Rachanavij/IR_Project.git)

2. Install dependencies:

```
python -m venv venv
venv\Scripts\activate
```

```
pip install -r requirements.txt
```

3. Crawl pages:

```
python run.py
```

4. Build index:

```
python indexer.py
```

5. Run Flask server:

```
python app.py
```

6. Submit CSV queries:

```
curl -X POST -F "file=@queries.csv" http://127.0.0.1:5000/search
```

7. Output: `results.csv` with top-K rankings.

## 11. Test Cases

- **Framework:** pytest
- **Modules Tested:** Crawler, Indexer, Processor
- **Execution:**

```
python -m pytest
```

- Tests include file generation, TF-IDF correctness, and query ranking validation.

```
(venv) C:\Users\vrach\Downloads\Fall25\IR\Project>python -m pytest
```

```
===== test session starts =====
```

```
platform win32 -- Python 3.13.9, pytest-9.0.2, pluggy-1.6.0
```

```
rootdir: C:\Users\vrach\Downloads\Fall25\IR\Project
```

```
collected 5 items
```

```
tests\test_crawler.py ..
```

```
tests\test_indexer.py ..
```

```
tests\test_processor.py .
```

```
===== 5 passed in 1.80s =====
```

```
(venv) C:\Users\vrach\Downloads\Fall25\IR\Project>
```

## 12. Limitations

- Limited to 10 crawled pages.
- Ranking is only TF-IDF cosine similarity.
- Optional NLP enhancements not implemented.
- Does not handle very large datasets efficiently.

## 13. Future Work

- Crawl hundreds/thousands of pages.
- Implement FAISS-based semantic search with embeddings.
- Integrate query expansion and spell correction.
- Deploy with a production-ready WSGI server.

## 14. Data Sources

- Wikipedia: [Data Science](#)
- Scrapy: <https://docs.scrapy.org>
- scikit-learn: <https://scikit-learn.org>

## 15. Bibliography

1. Salton, G., & McGill, M. J. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.
2. Manning, C. D., Raghavan, P., & Schütze, H. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
3. Goodfellow, I., Bengio, Y., & Courville, A. *Deep Learning*. MIT Press, 2016.
4. Hastie, T., Tibshirani, R., & Friedman, J. *The Elements of Statistical Learning*. Springer, 2009.
5. Wikipedia — [Data Science](#)
6. *Open AI*, ChatGPT.