Abstract

This project aimed to create a focused web crawler for information retrieval. The crawler targets Wikipedia articles and creates a basic document index to help with information retrieval.

Development Summary: The crawler uses the Scrapy framework to navigate and download Wikipedia articles. The downloaded pages are parsed with BeautifulSoup to extract text content. The extracted text goes through cleaning steps to remove irrelevant information and prepare it for further processing. Finally, a TF-IDF vectorizer and cosine similarity are used to generate a document index that records the relationships between crawled articles.

Objectives:The primary goal was to design and implement a web crawler capable of navigating Wikipedia and creating a basic document index. This index helps users find relevant articles based on content similarity.

Next steps: Future improvements could include extending the crawling scope beyond Wikipedia.
Implementing user queries and retrieval of relevant documents.
Improving the text cleaning process for better information extraction.
Investigate advanced indexing techniques such as Latent Semantic Indexing (LSI).

Overview

This project creates a framework for a focused crawler that uses Wikipedia for information retrieval tasks.

Solution Outline: The solution makes use of Scrapy's web scraping capabilities. BeautifulSoup helps you parse the downloaded HTML content and extract relevant text. Text cleaning techniques get the extracted text ready for further processing. Finally, TF-IDF and cosine similarity generate a document index that captures the semantic relationships between articles.

Relevant Literature: The project is based on concepts from web scraping using frameworks such as Scrapy [reference Scrapy documentation]. It also uses Natural Language Processing (NLP) techniques such as text cleaning and TF-IDF, which are commonly used in information retrieval tasks.

The proposed system is a specialized web crawler designed specifically for

Wikipedia. It retrieves and indexes documents, which allows users to find articles.

Design

The design focuses on crawling, text extraction, and document indexing.

System Capabilities:

Crawling Wikipedia articles to a specific depth limit.
Extracting text from downloaded HTML pages.
Cleaning the extracted text to remove any unnecessary information.
Creating a document index with TF-IDF and cosine similarity for document retrieval.
Interactions: The system uses Wikipedia to download articles. It also interacts with the user via an interface (to be developed in subsequent steps) that accepts queries and returns retrieved documents.

Scrapy for web scraping, BeautifulSoup for HTML parsing, re libraries for text cleaning, and scikit-learn for TF-IDF and cosine similarity calculations are all integrated into the system.

Architecture

The architecture is made up of multiple software components that work together to achieve desired functionality.

Components: Scrapy Spider: Manages crawling logic, links, and webpage downloads.
Parser uses BeautifulSoup to extract text content from downloaded HTML.
Text Cleaner uses regular expressions to remove HTML tags, escape characters, and irrelevant symbols.
Indexer: Uses scikit-learn's TF-IDF vectorizer to generate a document-term matrix and compute cosine similarities between documents.
Interfaces: The current system communicates with Wikipedia as its data source. Future development will include designing a user interface to accept queries and display retrieved documents.
The code makes use of Python libraries such as Scrapy, BeautifulSoup, re, and scikit-learn. The extracted data is saved in a pickled file for later use.

Operation

Install Python and Install Linux in windows

wsl –install

Install required libraries

Pip install scrapy

Pip install sckit-learn

pip install beautifulsoup4

pip install flask

pip install requests

Installation:

Scrapy Crawl:

Navigate to the spiders folder in the terminal.

Run the command: Scrapy crawl <file name> (replace <file name> with the actual spider file name).

This step calculates TF-IDF scores and cosine similarity for the HTML documents, storing the results in an index.pkl file.

Access the Index:

Go to the access pickle folder in the terminal.

Run the Python file to display the content of the index.pkl file.

Start Flask Server:

Navigate to the Flask folder in the terminal.

Run the Python file to initiate the Flask server.

Make a Query Request:

Open a new terminal.

Send a request to the Flask server with a query in the following format:

curl -X POST http://localhost:5000/query -H "Content-Type: application/json" -d '{"query": "python introduction"}'

The server will respond with a JSON format containing cosine similarity and document names of the top-k results

Conclusion

Success:

The crawler successfully crawls Wikipedia articles within the specified depth limit and builds a document index file (index.pkl) containing information about each crawled article, including its TF-IDF vector and cosine similarities with other articles.

Failure:

The script might encounter errors if:

Required libraries are not installed.

Wikipedia or the starting URL becomes inaccessible.

There are issues with parsing specific HTML structures.

Outputs:

Upon successful completion, a pickled file (index.pkl) is created, storing the built document index.

Caveats/Cautions:

The current crawling depth is limited . Adjust it cautiously to avoid overwhelming Wikipedia's servers.

The script focuses on text content and doesn't consider other elements like infoboxes or images, which might be relevant for information retrieval.

Data Sources

Scrapy - https://docs.scrapy.org/en/latest/

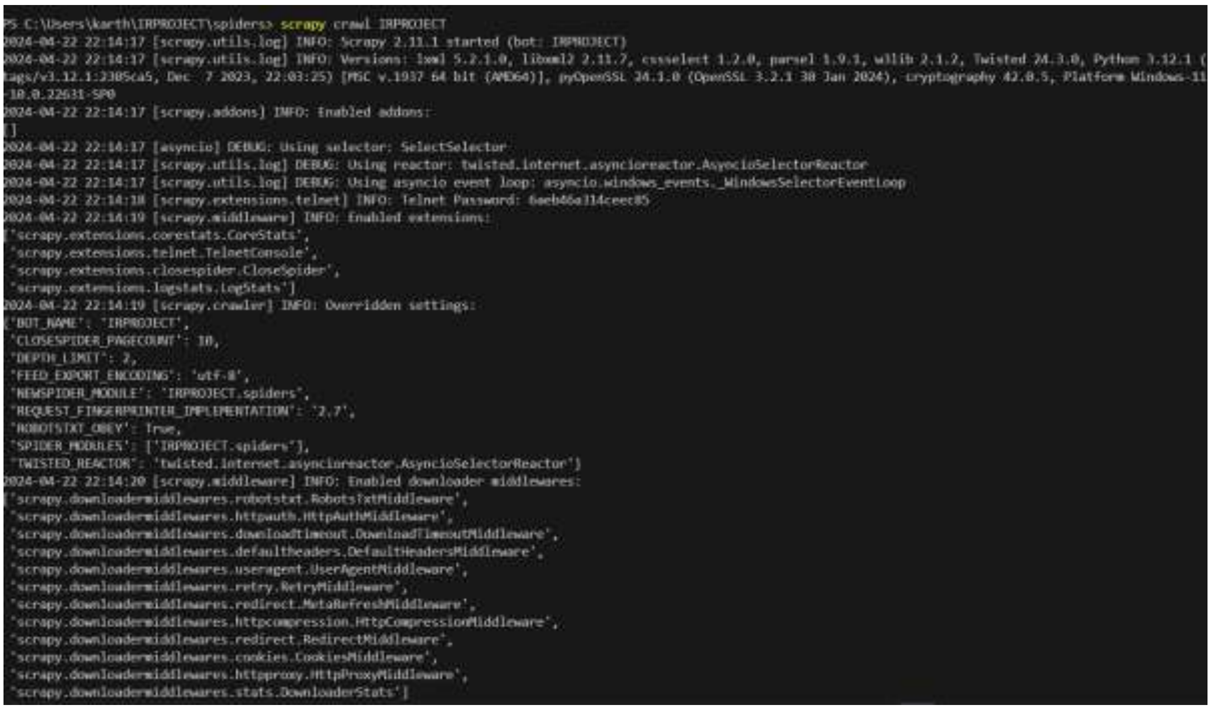
Beautiful Soup - https://beautiful-soup-4.readthedocs.io/en/latest/

Scikit-learn - scikit-learn: machine learning in Python — scikit-learn 1.4.2 documentation
Flask - https://flask.palletsprojects.com/en/3.0.x/

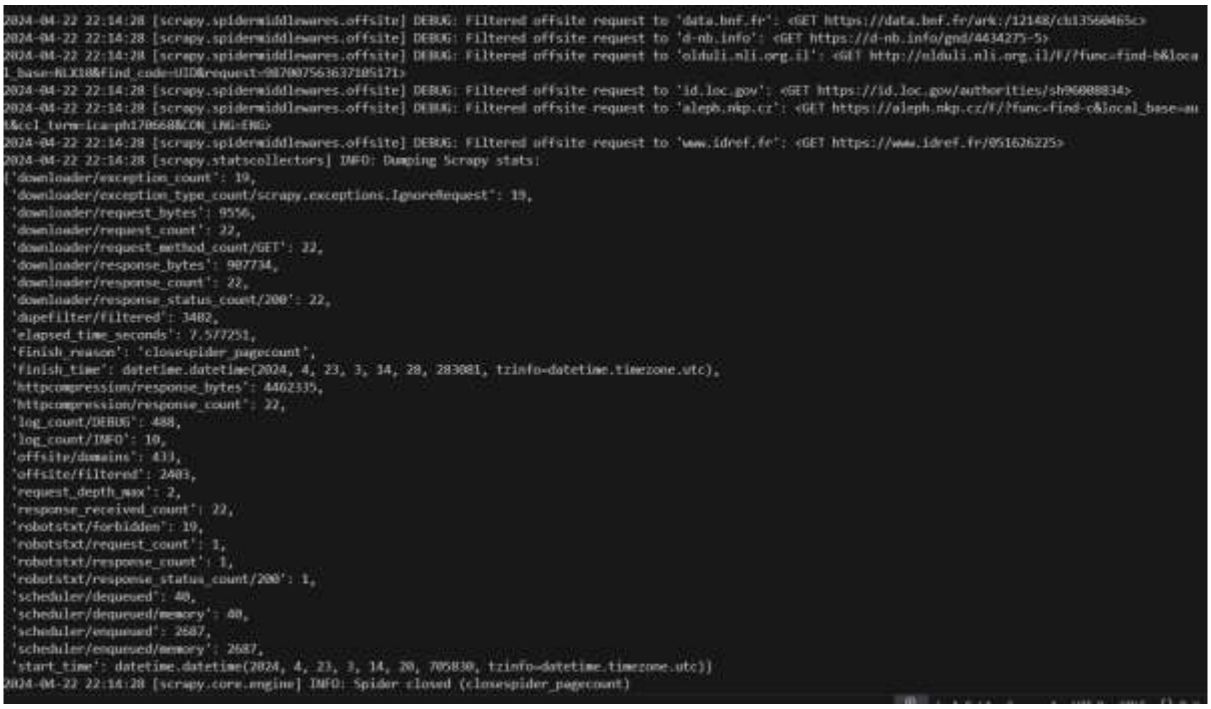
Source Code

Spider file - react_beauty_spider.py
Scrapped pages - react-{id}
Processor logic - processor.py
Query logic - query.py
Pickle file - tf-idf.pkl
Programmatic API - manual-request.py
Scikit API - scikit-request.py

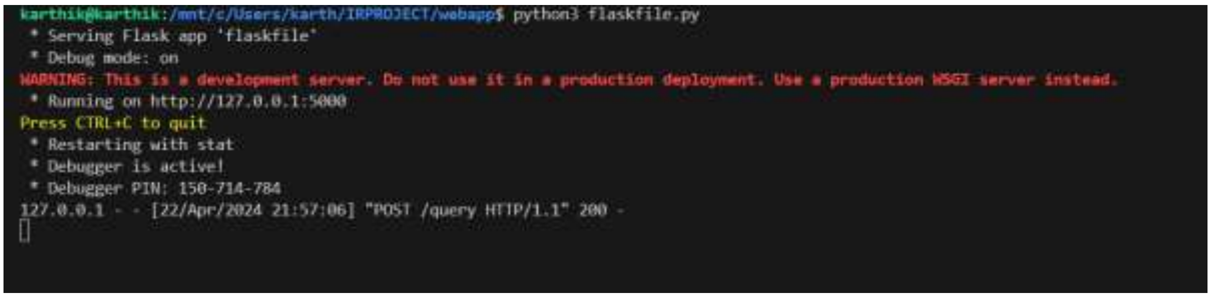
Test Cases:



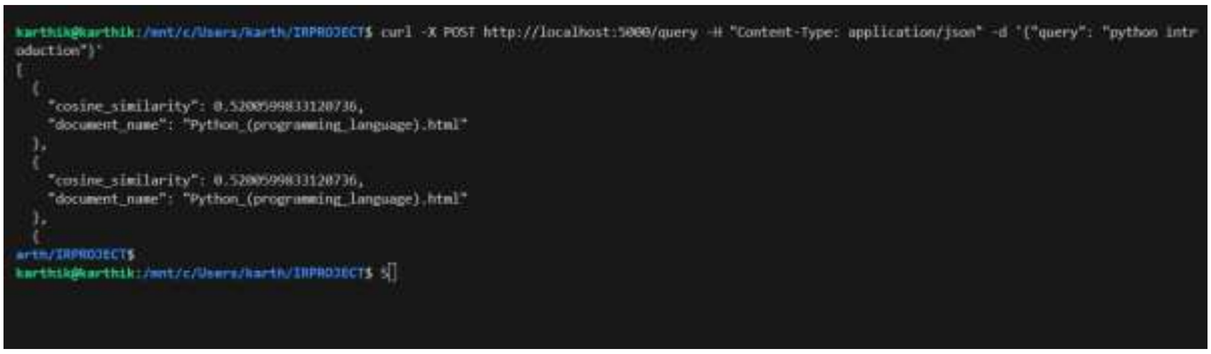
```
PS C:\Users\karth\IRPROJECT\spiders> scrapy crawl IRPROJECT
2024-04-22 22:14:17 [scrapy.utils.log] INFO: Scrapy 2.11.1 started (bot: IRPROJECT)
2024-04-22 22:14:17 [scrapy.utils.log] INFO: Versions: lxml 5.2.1.0, libxml2 2.11.7, cssselect 1.2.0, parsel 1.9.1, w3lib 2.1.2, Twisted 24.3.0, Python 3.12.1 (
tags/v3.12.1:2305ca5, Dec  7 2023, 22:03:25) [MSC v.1937 64 bit (AMD64)], pyOpenSSL 24.1.0 (OpenSSL 3.2.1 30 Jan 2024), cryptography 42.0.5, Platform Windows-11
-10.0.22631-SP0
2024-04-22 22:14:17 [scrapy.addons] INFO: Enabled addons:
[]
2024-04-22 22:14:17 [asyncio] DEBUG: Using selector: SelectSelector
2024-04-22 22:14:17 [scrapy.utils.log] DEBUG: Using reactor: twisted.internet.asyncioreactor.AsyncioSelectorReactor
2024-04-22 22:14:17 [scrapy.utils.log] DEBUG: Using asyncio event loop: asyncio.windows_events._WindowsSelectorEventLoop
2024-04-22 22:14:18 [scrapy.extensions.telnet] INFO: Telnet Password: 6aeb46a114ceec85
2024-04-22 22:14:19 [scrapy.middleware] INFO: Enabled extensions:
['scrapy.extensions.corestats.CoreStats',
 'scrapy.extensions.telnet.TelnetConsole',
 'scrapy.extensions.closespider.CloseSpider',
 'scrapy.extensions.logstats.LogStats']
2024-04-22 22:14:19 [scrapy.crawler] INFO: Overridden settings:
{'BOT_NAME': 'IRPROJECT',
 'CLOSESPIDER_PAGECOUNT': 10,
 'DEPTH_LIMIT': 2,
 'FEED_EXPORT_ENCODING': 'utf-8',
 'NEWSPIDER_MODULE': 'IRPROJECT.spiders',
 'REQUEST_FINGERPRINTER_IMPLEMENTATION': '2.7',
 'ROBOTSTXT_OBEY': True,
 'SPIDER_MODULES': ['IRPROJECT.spiders'],
 'TWISTED_REACTOR': 'twisted.internet.asyncioreactor.AsyncioSelectorReactor'}
2024-04-22 22:14:20 [scrapy.middleware] INFO: Enabled downloader middlewares:
['scrapy.downloadermiddlewares.robotstxt.RobotsTxtMiddleware',
 'scrapy.downloadermiddlewares.httpauth.HttpAuthMiddleware',
 'scrapy.downloadermiddlewares.downloadtimeout.DownloadTimeoutMiddleware',
 'scrapy.downloadermiddlewares.defaultheaders.DefaultHeadersMiddleware',
 'scrapy.downloadermiddlewares.useragent.UserAgentMiddleware',
 'scrapy.downloadermiddlewares.retry.RetryMiddleware',
 'scrapy.downloadermiddlewares.redirect.MetaRefreshMiddleware',
 'scrapy.downloadermiddlewares.httpcompression.HttpCompressionMiddleware',
 'scrapy.downloadermiddlewares.redirect.RedirectMiddleware',
 'scrapy.downloadermiddlewares.cookies.CookiesMiddleware',
 'scrapy.downloadermiddlewares.httpproxy.HttpProxyMiddleware',
 'scrapy.downloadermiddlewares.stats.DownloaderStats']
```



```
2024-04-22 22:14:28 [scrapy.spidermiddlewares.offsite] DEBUG: Filtered offsite request to 'data.bnf.fr': <GET https://data.bnf.fr/ark:/12148/cb13560465c>
2024-04-22 22:14:28 [scrapy.spidermiddlewares.offsite] DEBUG: Filtered offsite request to 'd-nb.info': <GET https://d-nb.info/gnd/4434275-5>
2024-04-22 22:14:28 [scrapy.spidermiddlewares.offsite] DEBUG: Filtered offsite request to 'olduli.nli.org.il': <GET http://olduli.nli.org.il/F/?func=find-b&loca
l_base=NLX10&find_code=UID&request=987007563637105171>
2024-04-22 22:14:28 [scrapy.spidermiddlewares.offsite] DEBUG: Filtered offsite request to 'id.loc.gov': <GET https://id.loc.gov/authorities/sh96008834>
2024-04-22 22:14:28 [scrapy.spidermiddlewares.offsite] DEBUG: Filtered offsite request to 'aleph.nkp.cz': <GET https://aleph.nkp.cz/F/?func=find-c&local_base=au
t&ccl_term=ica=ph170668&CON_LNG=ENG>
2024-04-22 22:14:28 [scrapy.spidermiddlewares.offsite] DEBUG: Filtered offsite request to 'www.idref.fr': <GET https://www.idref.fr/051626225>
2024-04-22 22:14:28 [scrapy.statscollectors] INFO: Dumping Scrapy stats:
{'downloader/exception_count': 19,
 'downloader/exception_type_count/scrapy.exceptions.IgnoreRequest': 19,
 'downloader/request_bytes': 9556,
 'downloader/request_count': 22,
 'downloader/request_method_count/GET': 22,
 'downloader/response_bytes': 907734,
 'downloader/response_count': 22,
 'downloader/response_status_count/200': 22,
 'dupefilter/filtered': 3482,
 'elapsed_time_seconds': 7.577251,
 'finish_reason': 'closespider_pagecount',
 'finish_time': datetime.datetime(2024, 4, 23, 3, 14, 28, 283081, tzinfo=datetime.timezone.utc),
 'httpcompression/response_bytes': 4462335,
 'httpcompression/response_count': 22,
 'log_count/DEBUG': 488,
 'log_count/INFO': 10,
 'offsite/domains': 433,
 'offsite/filtered': 2403,
 'request_depth_max': 2,
 'response_received_count': 22,
 'robotstxt/forbidden': 19,
 'robotstxt/request_count': 1,
 'robotstxt/response_count': 1,
 'robotstxt/response_status_count/200': 1,
 'scheduler/dequeued': 40,
 'scheduler/dequeued/memory': 40,
 'scheduler/enqueued': 2687,
 'scheduler/enqueued/memory': 2687,
 'start_time': datetime.datetime(2024, 4, 23, 3, 14, 20, 705830, tzinfo=datetime.timezone.utc)}
2024-04-22 22:14:28 [scrapy.core.engine] INFO: Spider closed (closespider_pagecount)
```



```
karthik@karthik:/mnt/c/Users/karth/IRPROJECT/webapp$ python3 flaskfile.py
 * Serving Flask app 'flaskfile'
 * 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: 150-714-784
127.0.0.1 - - [22/Apr/2024 21:57:06] "POST /query HTTP/1.1" 200 -
```



```
karthik@karthik:/mnt/c/Users/karth/IRPROJECT$ curl -X POST http://localhost:5000/query -H "Content-Type: application/json" -d '{"query": "python intr
oduction"}'
[
  {
    "cosine_similarity": 0.5200599833120736,
    "document_name": "Python_(programming_language).html"
  },
  {
    "cosine_similarity": 0.5200599833120736,
    "document_name": "Python_(programming_language).html"
  },
  {
arth/IRPROJECT$
karthik@karthik:/mnt/c/Users/karth/IRPROJECT$ $
```

Source Code

```
import scrapy
from pathlib import Path
from bs4 import BeautifulSoup
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

```python
import pickle
import re
class WikipediaCrawlerSpider(scrapy.Spider):
    name = 'IRPROJECT'
    allowed_domains = ['en.wikipedia.org']
    start_urls = ['https://en.wikipedia.org/wiki/Python_(programming_language)']

    custom_settings = {
        'DEPTH_LIMIT': 2,
        'CLOSESPIDER_PAGECOUNT': 10,  # Set maximum page count
    }
    page_count = 0
    documents = []
    document_names = []

    def parse(self, response):
        self.page_count += 1
        filename = self.get_valid_filename(response.url.split("/")[-1]) + '.html'
        self.document_names.append(filename)  # Store document name
        if self.page_count > self.settings.get('CLOSESPIDER_PAGECOUNT'):
            return
        if response.meta['depth'] > self.settings.get('DEPTH_LIMIT'):
            return
        with open(filename, "wb") as file:
            file.write(response.body)
        self.log(f"Saved file {filename}")

        with open(filename, "rb") as file:
            # Decode bytes to string using utf-8 encoding
            html_content = file.read().decode('utf-8')
            soup = BeautifulSoup(html_content, 'html.parser')
            text = soup.get_text()
            # Clean the text
            clean_text = self.clean_text(text)
            self.documents.append(clean_text)

        for next_page in response.css('a::attr(href)').getall():
            yield response.follow(next_page, callback=self.parse)

    def closed(self, reason):
        self.build_index()

    def build_index(self):
        vectorizer = TfidfVectorizer()
        tfidf_matrix = vectorizer.fit_transform(self.documents)
        cosine_sim_matrix = cosine_similarity(tfidf_matrix, tfidf_matrix)

        index = {}
        for idx, (doc_name, doc) in enumerate(zip(self.document_names,
self.documents)):
            index[idx] = {
                'document_name': doc_name,
                'document': doc,
                'tfidf_vector': tfidf_matrix[idx],
                'cosine_similarities': cosine_sim_matrix[idx]
            }

        with open('index.pkl', 'wb') as file:
            pickle.dump(index, file)

    def get_valid_filename(self, filename):
        # Remove invalid characters from filename
        return re.sub(r'[<>:"/\\|?*]', '_', filename)

    def clean_text(self, text):
        # Remove HTML tags
        clean_text = re.sub(r'<.*?>', '', text)
        # Remove escape characters
        clean_text = re.sub(r'\\[ntr]', '', clean_text)
        # Remove other special symbols
        clean_text = re.sub(r'[^a-zA-Z0-9\s]', '', clean_text)
        # Remove extra whitespaces
        clean_text = re.sub(r'\s+', ' ', clean_text)
        # Convert to lowercase
        clean_text = clean_text.lower()
        return clean_text
```

FLASK CODE :

```
import os

import pickle
from flask import Flask, jsonify, request
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

```
# Determine the path to index.pkl
index = os.path.join(os.path.dirname(__file__), '..' ,'spiders', 'index.pkl')

# Load the index.pkl file
with open(index, 'rb') as f:
    index = pickle.load(f)

# Continue with your Flask app setup
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform([doc['document'] for doc in
index.values()])

Webapp = Flask(__name__)

@Webapp.route('/query', methods=['POST'])
def query():
    queryJson = request.json
    query = queryJson.get('query', '')

    Vector = vectorizer.transform([query])
    cosine_similarities = cosine_similarity(Vector, tfidf_matrix).flatten()
   #change k value here to get top k results
    k = min(5, len(cosine_similarities))
    top_indices_of_k = cosine_similarities.argsort()[-k:][::-1]

    results = [{'cosine_similarity': cosine_similarities[idx],
                'document_name': index[idx]['document_name'],} for idx
in  top_indices_of_k]
    return jsonify(results)


if __name__ == '__main__':
   Webapp.run(debug=True)
```

Bibliography

https://requests.readthedocs.io/en/latest/

https://flask.palletsprojects.com/en/3.0.x/

https://scikit-learn.org/stable/

https://scrapy.org/