

1: Scrapy Crawler  
Done with assistance from ChatGPT (bug fixing and explanation)  
Scrapy Tutorial — Scrapy 2.3.0 documentation. (n.d.). Docs.scrapy.org. <https://docs.scrapy.org/en/latest/intro/tutorial.html>

```
In [ ]: import scrapy, os
from scrapy.linkextractors import LinkExtractor
from urllib.parse import urlparse

class HtmlCrawler(scrapy.Spider):
    name = "html_crawler"
    # initialize crawler
    def __init__(self, seed="https://quotes.toscrape.com/", max_pages=20, max_depth=2, *args, **kwargs):
        super(HtmlCrawler, self).__init__(*args, **kwargs)
        self.start_urls = [seed]
        self.allowed_domain = urlparse(seed).netloc
        self.max_pages = int(max_pages)
        self.max_depth = int(max_depth)
        self.page_count = 0
        self.link_extractor = LinkExtractor(allow_domains=[self.allowed_domain])

    def parse(self, response):
        # stop if reached max pages
        if self.page_count >= self.max_pages:
            self.crawler.engine.close_spider(self, reason="max pages reached")
            return

        # increment and get page url and text
        self.page_count += 1
        pageURL = response.url
        pageHTML = response.text

        # save content to file
        filename = f"downloaded_page_{self.page_count}.html"
        path = os.path.join("../", filename)

        with open(path, "w", encoding="utf-8") as f:
            f.write(pageHTML)
        self.logger.info(f"Saved {pageURL} as {filename}")

        # get next link if within depth
        currDepth = response.meta.get('depth', 0)
        if currDepth < self.max_depth:
            for link in self.link_extractor.extract_links(response):
                yield scrapy.Request(link.url, callback=self.parse, meta={'depth': currDepth + 1})
```

2: Scikit-learn Indexer  
Done with assistance from ChatGPT (bug fixing and explanation)  
Examples. (2025). Scikit-Learn. [https://scikit-learn.org/stable/auto\\_examples/Working-With-Text-Data-\(2024\).Scikit-Learn-https://scikit-learn.org/1-4/tutorial/text\\_analytics/working\\_with\\_text\\_data.html](https://scikit-learn.org/stable/auto_examples/Working-With-Text-Data-(2024).Scikit-Learn-https://scikit-learn.org/1-4/tutorial/text_analytics/working_with_text_data.html)

```
In [ ]: # Project assisted and partially generated using the assistance of LLM ChatGPT
import os
import json
import glob
import re
from bs4 import BeautifulSoup
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

class SklearnIndexer:
    def __init__(self, input_folder="", output_file="./inverted_index.json"):
        self.input_folder = input_folder
        self.output_file = output_file
        self.documents = []
        self.doc_ids = []
        self.vectorizer = None
        self.tfidf_matrix = None
        self.feature_names = None

    # html to plaintext
    def html_to_text(self, html):
        soup = BeautifulSoup(html, "html.parser")
        text = soup.get_text(separator=" ", strip=True)
        splitText = re.sub(r"\s+", " ", text)
        return splitText

    # load the downloaded files
    def load_documents(self):
        files = glob.glob(os.path.join(self.input_folder, "downloaded_page_*.html"))

        for path in files:
            with open(path, "r", encoding="utf-8", errors="ignore") as f:
                html = f.read()

                text = self.html_to_text(html)

                self.documents.append(text)
                self.doc_ids.append(os.path.basename(path))

        print(f"Loaded {len(self.documents)} documents.")

    # build tfidf matrix
    def build_tfidf(self):
        self.vectorizer = TfidfVectorizer(
            stop_words="english",
            lowercase=True,
        )

        self.tfidf_matrix = self.vectorizer.fit_transform(self.documents)
        self.feature_names = self.vectorizer.get_feature_names_out()

    # build the inverted index
    def build_inverted_index(self):
        inverted = {}

        rows, cols = self.tfidf_matrix.nonzero()

        for row, col in zip(rows, cols):
            term = self.feature_names[col]
            score = float(self.tfidf_matrix[row, col])
            docID = self.doc_ids[row]

            if term not in inverted:
                inverted[term] = []

            inverted[term].append({
                "doc_id": docID,
                "tfidf": score
            })

        return inverted

    # save index to a json file
    def save_index(self, inverted_index):
        os.makedirs(os.path.dirname(self.output_file), exist_ok=True)

        with open(self.output_file, "w", encoding="utf-8") as f:
            json.dump(inverted_index, f, indent=4)

        print(f"Saved index at {self.output_file}")

    # load docs, build tfidf, build index
    def run(self):
        self.load_documents()
        self.build_tfidf()
        inverted = self.build_inverted_index()
        self.save_index(inverted)

    # find based on cosine similarity
    def search(self, query, top_k=5):
        # create query vec and find cosine similarity
        queryVec = self.vectorizer.transform([query])
        scores = cosine_similarity(queryVec, self.tfidf_matrix)[0]

        # return sorted by rank
        ranked = sorted(zip(self.doc_ids, scores), key=lambda x: x[1], reverse=True)

        return ranked[:top_k]
```

Example Output

```
In [ ]: "tea": [
    {
        "doc_id": "downloaded_page_1.html",
        "tfidf": 0.06385495787246827
    },
    {
        "doc_id": "downloaded_page_10.html",
        "tfidf": 0.18246253148246371
    },
    {
        "doc_id": "downloaded_page_15.html",
        "tfidf": 0.1486372739369191
    },
    {
        "doc_id": "downloaded_page_3.html",
        "tfidf": 0.09926167534888873
    },
    {
        "doc_id": "downloaded_page_9.html",
        "tfidf": 0.13222187825641787
    }
],
"bag": [
    {
        "doc_id": "downloaded_page_1.html",
        "tfidf": 0.08358233517458886
    },
    {
        "doc_id": "downloaded_page_15.html",
        "tfidf": 0.18390961686661183
    }
],
"know": [
    {
        "doc_id": "downloaded_page_1.html",
        "tfidf": 0.09875103357093636
    },
    {
        "doc_id": "downloaded_page_11.html",
        "tfidf": 0.04263865531418255
    },
    {
        "doc_id": "downloaded_page_12.html",
        "tfidf": 0.1868937162278184
    },
    {
        "doc_id": "downloaded_page_13.html",
        "tfidf": 0.12817676879799435
    },
    {
        "doc_id": "downloaded_page_14.html",
        "tfidf": 0.03329951675649862
    },
    {
        "doc_id": "downloaded_page_15.html",
        "tfidf": 0.10874784686765194
    },
    {
        "doc_id": "downloaded_page_2.html",
        "tfidf": 0.12817676879799435
    },
    {
        "doc_id": "downloaded_page_7.html",
        "tfidf": 0.0603104284787756
    },
    {
        "doc_id": "downloaded_page_9.html",
        "tfidf": 0.10223988538881692
    }
],
]
```

3: Flask Application and Query Handling  
Done with assistance from ChatGPT (bug fixing and explanation) Flask. (2010). Welcome to Flask — Flask Documentation (3.0.x). Palletsprojects.com. <https://flask.palletsprojects.com/en/stable/> NLTK : ntk. (n.d.). Www.nltk.org. [https://www.nltk.org/\\_modules/nltk.html](https://www.nltk.org/_modules/nltk.html)

```
In [ ]: # Project assisted and partially generated using the assistance of LLM ChatGPT
from flask import Flask, request, Response
import csv
from indexer import SklearnIndexer
import os
from nltk.metrics import edit_distance

app = Flask(__name__)
INDEX_FILE = os.path.join(".", "inverted_index.json")
DOC_FOLDER = "."

indexer = SklearnIndexer(input_folder=DOC_FOLDER, output_file=INDEX_FILE)
VOCAB = set()

# run indexer if not existant
if not os.path.exists(INDEX_FILE):
    print("Inverted index not found, running indexer")
    indexer.run()
    VOCAB = set(indexer.vectorizer.get_feature_names_out())
else:
    print("Loading existing index")
    indexer.load_documents()
    indexer.build_tfidf()
    VOCAB = set(indexer.vectorizer.get_feature_names_out())

# spelling correction using edit distance
def spelling_check(query, vocab, max_distance=2):
    tokens = query.lower().split()
    suggestions = []

    # go through each token and find closest word using edit distance
    for t in tokens:
        if len(t) <= 2:
            continue

        closestWord = None
        closestDist = 999

        for v in vocab:
            d = edit_distance(t, v)
            if d < closestDist:
                closestDist = d
                closestWord = v
            if closestDist == 0:
                break

        if closestDist <= max_distance and closestWord != t:
            suggestions[t] = closestWord

    return suggestions if suggestions else None

# format output
def format_output(query, data):
    lines = []
    lines.append(f"\nQuery: {query}")
    lines.append("Results:")
    for item in data.get("results", []):
        lines.append(f" {item['document']} -> {item['score']}")

    suggestions = data.get("suggestions")
    if suggestions:
        lines.append("Suggestions:")
        for wrong, correct in suggestions.items():
            lines.append(f" {wrong} -> {correct}")
    else:
        lines.append("Suggestions: None")

    return "\n".join(lines)

@app.route("/query", methods=["POST"])
def query_processor():
    if 'file' not in request.files:
        return Response("ERROR: No file provided", status=400)

    file = request.files['file']

    if not file.filename.endswith(".csv"):
        return Response("ERROR: Only CSV files allowed", status=400)

    decoded = file.read().decode("utf-8").splitlines()
    reader = csv.DictReader(decoded)

    if 'query' not in reader.fieldnames:
        return Response("ERROR: CSV must contain a 'query' column", status=400)

    queries = []
    for row in reader:
        text = row['query'].strip()
        if text:
            queries.append(text)

    if not queries:
        return Response("ERROR: No valid queries found", status=400)

    # build text output
    output_text = ""
    for q in queries:
        suggestion = spelling_check(q, VOCAB)
        raw_results = indexer.search(q, top_k=5)

        results = [{"document": doc, "score": float(score)}
                    for doc, score in raw_results]

        format_results = format_output(q, {"results": results, "suggestions": suggestion})
        output_text += format_results + "\n\n"

    return (output_text)
```

Test Flask Queries

```
In [ ]: import requests

csv_file_path = "queries.csv"

url = "http://127.0.0.1:5000/query"

with open(csv_file_path, "rb") as f:
    files = {"file": f}
    try:
        response = requests.post(url, files=files)
        response.raise_for_status()
    except requests.exceptions.RequestException as e:
        print(f"Request failed: {e}")
    else:
        print(response.text)
```

Query Search Example:

```
In [ ]: Query: quotes
Results:
downloaded_page_12.html -> 0.29662668165228943
downloaded_page_7.html -> 0.2546243690774928
downloaded_page_8.html -> 0.251562436948781
downloaded_page_5.html -> 0.2510613501789357
downloaded_page_2.html -> 0.2443986794998816
Suggestions: None

Query: stupid
Results:
downloaded_page_12.html -> 0.20758141045778497
downloaded_page_10.html -> 0.15898778532077524
downloaded_page_19.html -> 0.0829298060808952
downloaded_page_11.html -> 0.0539586447997341
downloaded_page_17.html -> 0.050748502876156345
Suggestions: None

Query: world
Results:
downloaded_page_1.html -> 0.12371686314979401
downloaded_page_16.html -> 0.10161315288356786
downloaded_page_13.html -> 0.06355193616728467
downloaded_page_19.html -> 0.047534828618043
downloaded_page_9.html -> 0.03428312693854897
Suggestions: None

Query: stupid1
Results:
downloaded_page_1.html -> 0.0
downloaded_page_10.html -> 0.0
downloaded_page_11.html -> 0.0
downloaded_page_12.html -> 0.0
downloaded_page_13.html -> 0.0
Suggestions:
stupid1 -> stupid

Query: lady
Results:
downloaded_page_12.html -> 0.19429842272174835
downloaded_page_10.html -> 0.1413856858538812
downloaded_page_19.html -> 0.0829298060808952
downloaded_page_11.html -> 0.0539586447997341
downloaded_page_17.html -> 0.047531593893433366
Suggestions: None

Query: ladyy
Results:
downloaded_page_1.html -> 0.0
downloaded_page_10.html -> 0.0
downloaded_page_11.html -> 0.0
downloaded_page_12.html -> 0.0
downloaded_page_13.html -> 0.0
Suggestions:
ladyy -> lady

Query: love
Results:
downloaded_page_9.html -> 0.366248826583218955
downloaded_page_14.html -> 0.233642804676565
downloaded_page_4.html -> 0.11286637354168547
downloaded_page_12.html -> 0.10369398741464176
downloaded_page_7.html -> 0.08901584461544956
Suggestions: None
```