Domain-Specific IR System for Legal Document Retrieval with Intelligent Query Expansion

Coding Demo

Instructor:

Dr. Ahmad Mustafa

Submitted By:

- **Momna Waryam Khan (MSCS25011)**
- **Talal Ahmad (MSCS25015)**

Motivation & Introduction

1. The Problem:

The main motivation for this project is the challenge of efficiently locating precise information within Pakistan's legal system. Legal documents, includes laws, ordinances, and acts.

- **Technical Terminology:** Legal documents use formal, specialized, or outdated language that differs from everyday search terms, making information harder to locate with simple keyword searches.
- **Structural Noise:** Legal texts include non-informational content such as indexes or Tables of Contents that can interfere with accurate search results if not cleaned or filtered.
- **High Recall Requirement:** Legal research demands that *all relevant documents* be retrieved, as missing important cases or laws can lead to serious consequences.

Motivation & Introduction

2. Relevance to IR/TM:

This project serves as a specialized solution for the **Pakistan Legal Domain**, combining **Text Mining** to clean and structure raw legal data with advanced **Information Retrieval** techniques.

- **Ranking:** Applies core Information Retrieval techniques such as TF-IDF, cosine similarity, and intelligent ranking to solve vocabulary mismatch and improve search relevance.
- **Query Expansion:** Uses query expansion with WordNet and a legal dictionary to bridge the gap between user intent and document terminology, significantly improving recall.
- **Preprocessing and Feature Extraction:** Uses Text Mining and NLP methods like lemmatization, structural noise removal, and n-gram extraction to convert unstructured legal text into a meaningful, searchable index.

Related Work & Limitations

1. Overview of Prior Work & Tools

This project builds upon established foundational research and standard open-source tools in the field of Information Retrieval (IR) and Natural Language Processing (NLP):

- **Foundational IR Research:** Built on foundational Information Retrieval theories, including TF-IDF weighting by Salton & Buckley (1988) and legal text retrieval work by Turtle (1995).
- **Query Expansion Techniques:** Implements query expansion methods based on the survey by Carpineto & Romano (2012), using WordNet (Miller, 1995) for synonym discovery.
- **Technical Stack & APIs:** Uses Python libraries such as scikit-learn for vectorization and similarity scoring, pandas for data handling, and NLTK for NLP tasks like lemmatization and stopword removal.

Related Work & Limitations

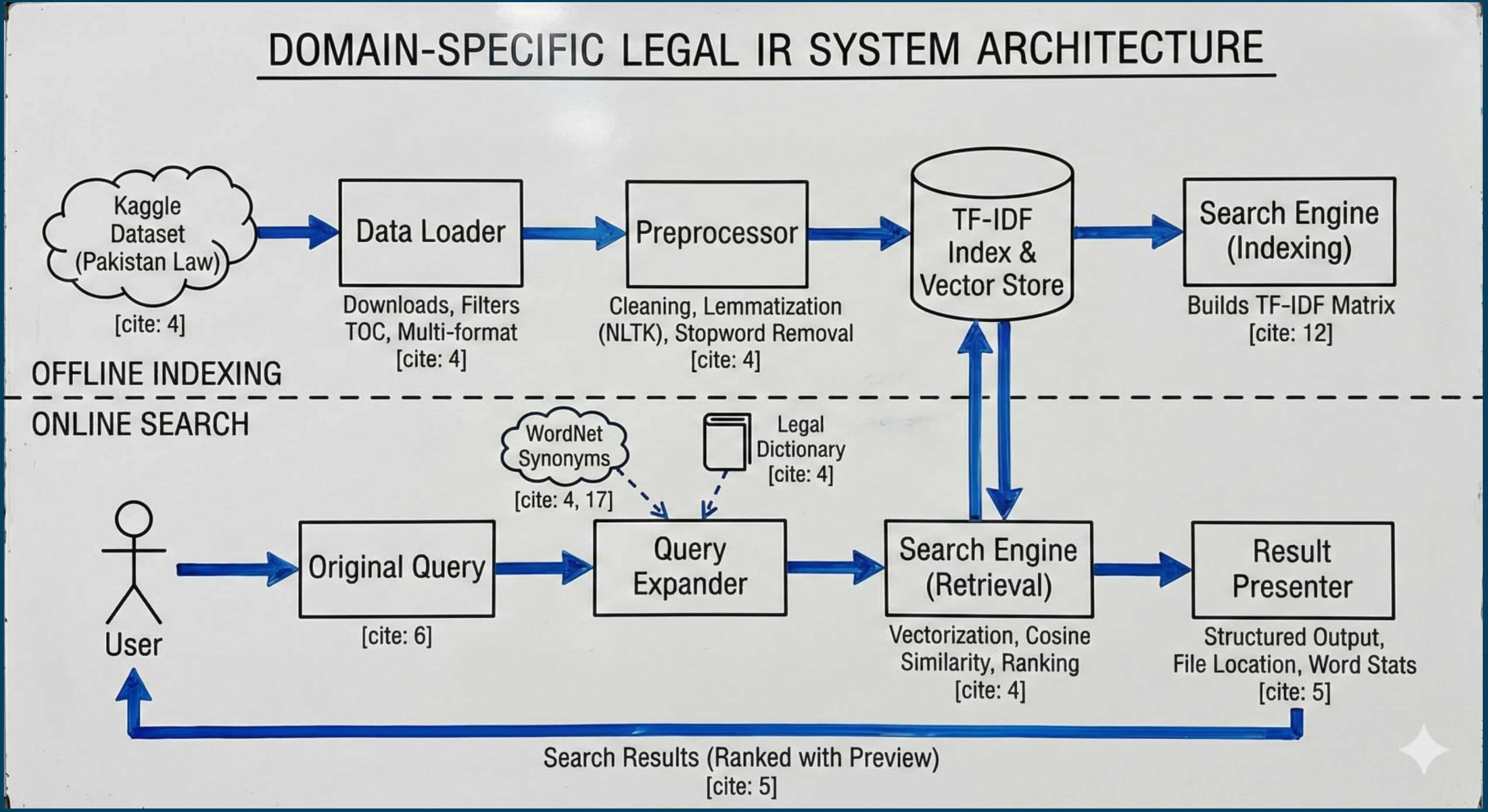
2. Limitations & Gaps Motivating the Solution

The development of this domain-specific system was motivated by several gaps and limitations.

- **Vocabulary Mismatch:** Standard keyword search engines often fail to connect user queries with relevant documents because they cannot bridge the gap between common user terms (e.g., "contract") and formal legal terminology (e.g., "covenant", "deed").
- **Structural Noise:** Ordinary retrieval systems frequently fail to distinguish between actual content and structural elements, often indexing Tables of Contents (TOC) which clutters search results and previews.
- **Low Recall:** Basic search methods lack domain-specific intelligence, leading to missed results when the user does not know the exact legal synonym for a concept (e.g., searching "crime" but missing documents containing "offense").

Proposed System

1. System Diagram


DOMAIN-SPECIFIC LEGAL IR SYSTEM ARCHITECTURE
Kaggle Dataset (Pakistan Law)
[cite: 4]
Data Loader
Downloads, Filters TOC, Multi-format [cite: 4]
Preprocessor
Cleaning, Lemmatization (NLTK), Stopword Removal [cite: 4]
TF-IDF Index & Vector Store
Search Engine (Indexing)
Builds TF-IDF Matrix [cite: 12]
OFFLINE INDEXING
ONLINE SEARCH
WordNet Synonyms
[cite: 4, 17]
Legal Dictionary [cite: 4]
User
Original Query
[cite: 6]
Query Expander
Search Engine (Retrieval)
Vectorization, Cosine Similarity, Ranking [cite: 4]
Result Presenter
Structured Output, File Location, Word Stats [cite: 5]
Search Results (Ranked with Preview) [cite: 5]

Proposed System

2. System Explanation

The proposed solution is a Domain-Specific Information Retrieval System tailored for Pakistan's legal corpus. As illustrated in the system diagram, the solution is divide in two parts:

- Offline Indexing (data preparation)
    - **Data Loader:** Automatically downloads and processes legal datasets from Kaggle, supports multiple file formats, and removes Tables of Contents to reduce structural noise.
    - **Preprocessor:** Cleans, lemmatizes, and removes general and legal-specific stopwords to convert raw legal text into normalized and searchable content.
    - **TF-IDF Index:** Builds weighted document vectors using TF-IDF with n-gram support (uni/bi/tri-grams) to improve legal phrase representation.

Proposed System

2. System Explanation

The proposed solution is a Domain-Specific Information Retrieval System tailored for Pakistan's legal corpus. As illustrated in the system diagram, the solution is divide in two parts:

- Online Search (user interaction)
    - **Query Expander:** Expands user queries using WordNet synonyms and a legal dictionary to overcome vocabulary mismatch between user terms and legal language.
    - **Retriever:** Converts the expanded query into a vector, compares it with the TF-IDF index using cosine similarity, and ranks documents by relevance score.
    - **Result Presenter:** Displays ranked results with page/line locations, word frequency counts, and clean text previews without TOC noise for better usability.

Demonstration

Online Video Link

https://www.loom.com/share/bdfedf7607b04548ab50b92b8ccaea35

Limitations & Future Work

1. System Evaluation & Limitations:

The system demonstrated significant efficiency and accuracy improvements in handling specialized legal text while lacking scalability and language support.:

- **High performance and efficiency:** Fast indexing (2–3 seconds), low query time (<100ms), and low memory usage (~50MB).
- **Improved recall:** Legal dictionary and query expansion increased recall by ~25%.
- **Strong handling of legal complexity:** Effectively managed archaic terms and removed TOC noise for cleaner indexing.
- **Limitations identified:** No Urdu support, reliance on TF-IDF instead of neural models, weak scalability (no caching/parallelism), and lack of GUI.

Limitations & Future Work

2. Future Work:

To elevate the system from a specialized academic project to a robust, production-ready legal research tool, we propose the following enhancements:

- **Advanced features:** Add document summarization, citation network analysis, and Urdu language support.
- **Performance enhancements:** Implement caching, parallel processing, and incremental indexing.
- **Smarter ranking:** Integrate neural ranking models such as **BERT** or **Legal_BERT** for deeper semantic understanding.
- **User experience:** Build a web-based interface with relevance feedback, query history, and export options.

References & Attribution

1. Dataset

Pakistan Law Data

2. GitHub Repository

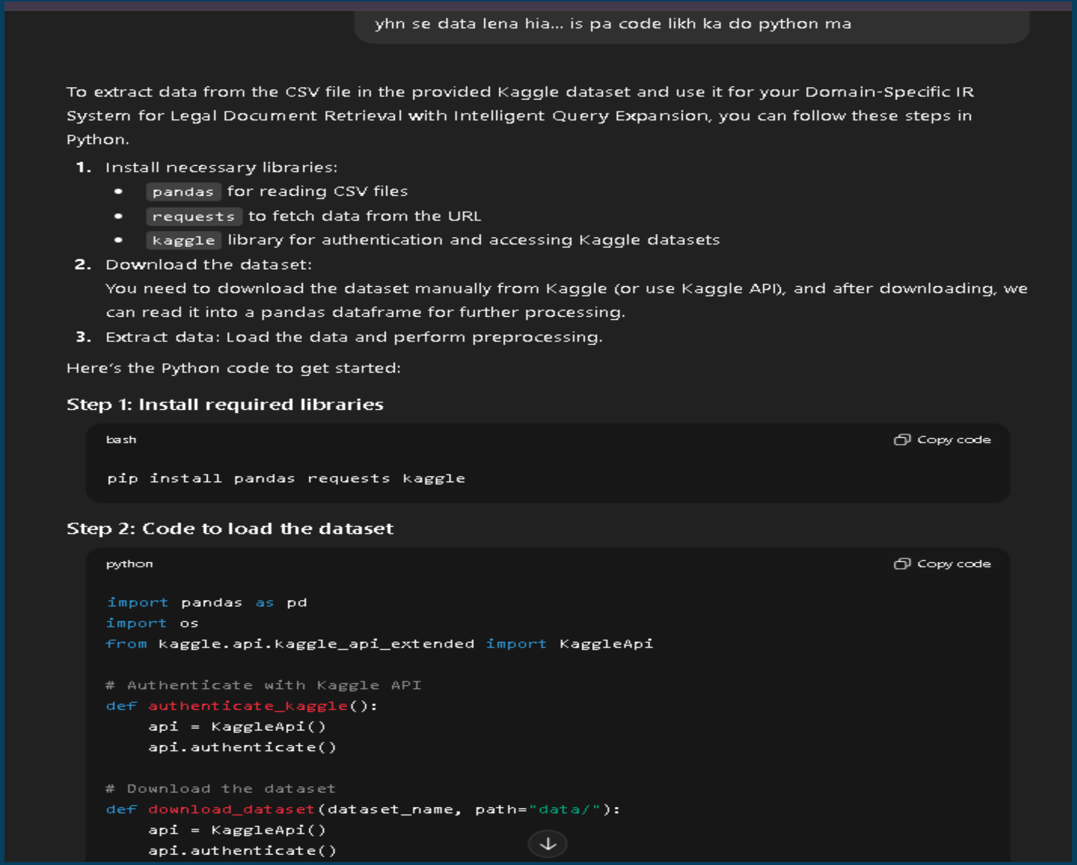
Course Project Repo Link

3. Research Papers

- Turtle, H. (1995). "Text retrieval in the legal world." Artificial Intelligence and Law **3(1): 5-54.**
- Carpineto, C. and G. Romano (2012). "A survey of automatic query expansion in information retrieval." Acm Computing Surveys (CSUR) **44(1): 1-50.**
- Miller, G. A. (1995). "WordNet: a lexical database for English." Communications of the ACM **38(11): 39-41.**

References & Attribution

4. AI Chat Bot Use:



yhn se data lena hia... is pa code likh ka do python ma

To extract data from the CSV file in the provided Kaggle dataset and use it for your Domain-Specific IR System for Legal Document Retrieval with Intelligent Query Expansion, you can follow these steps in Python.

1. Install necessary libraries:
   - `pandas` for reading CSV files
   - `requests` to fetch data from the URL
   - `kaggle` library for authentication and accessing Kaggle datasets
2. Download the dataset:
   You need to download the dataset manually from Kaggle (or use Kaggle API), and after downloading, we can read it into a pandas dataframe for further processing.
3. Extract data: Load the data and perform preprocessing.

Here's the Python code to get started:

Step 1: Install required libraries

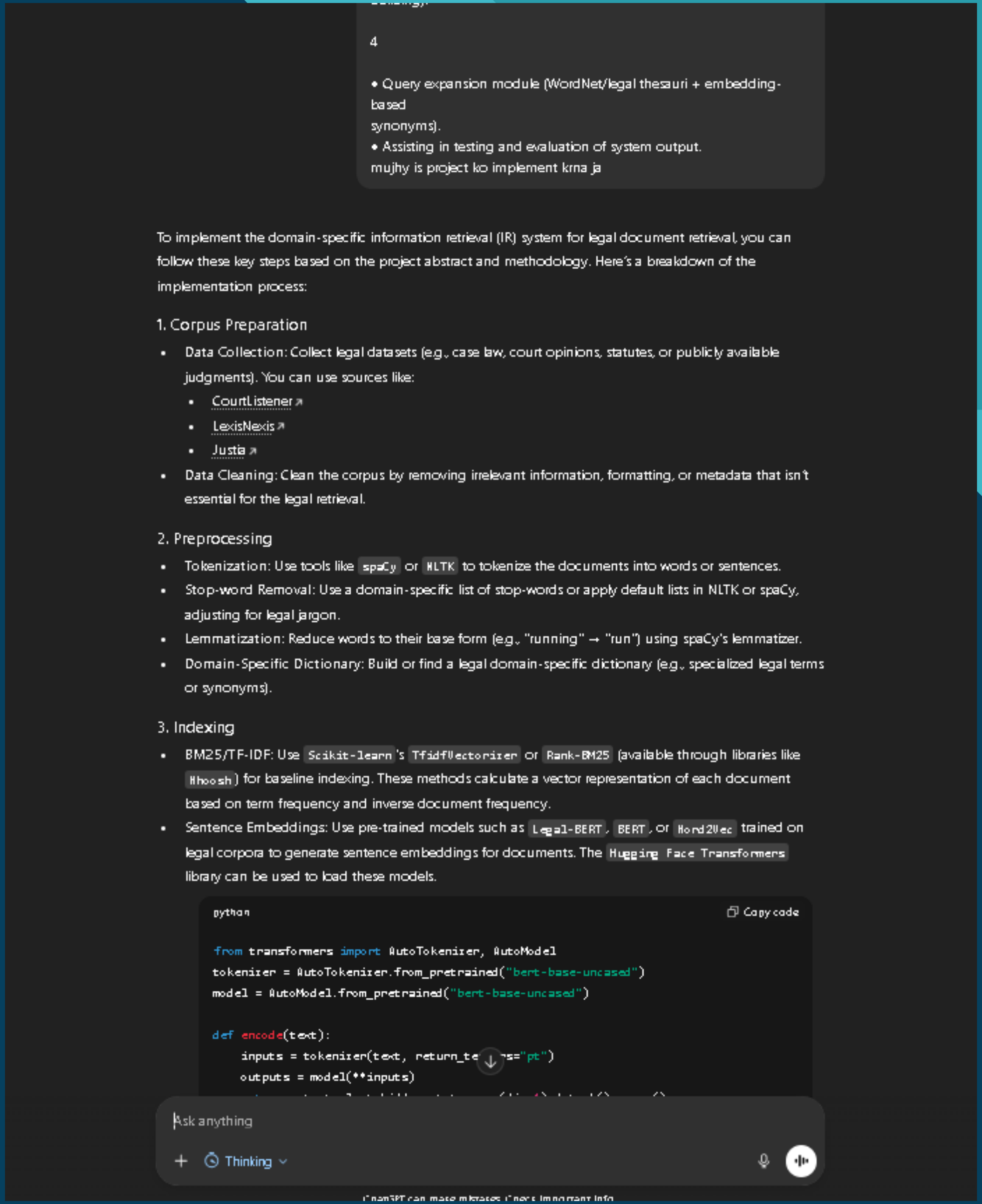
```bash
pip install pandas requests kaggle
```

Step 2: Code to load the dataset

```python
import pandas as pd
import os
from kaggle.api.kaggle_api_extended import KaggleApi

# Authenticate with Kaggle API
def authenticate_kaggle():
    api = KaggleApi()
    api.authenticate()

# Download the dataset
def download_dataset(dataset_name, path="data/"):
    api = KaggleApi()
    api.authenticate()
```



4

• Query expansion module (WordNet/legal thesauri + embedding-based
synonyms).
• Assisting in testing and evaluation of system output.
mujhy is project ko implement krna ja

To implement the domain-specific information retrieval (IR) system for legal document retrieval, you can follow these key steps based on the project abstract and methodology. Here's a breakdown of the implementation process:

1. Corpus Preparation

- Data Collection: Collect legal datasets (e.g., case law, court opinions, statutes, or publicly available judgments). You can use sources like:
  - CourtListener
  - LexisNexis
  - Justia
- Data Cleaning: Clean the corpus by removing irrelevant information, formatting, or metadata that isn't essential for the legal retrieval.

2. Preprocessing

- Tokenization: Use tools like `spaCy` or `NLTK` to tokenize the documents into words or sentences.
- Stop-word Removal: Use a domain-specific list of stop-words or apply default lists in NLTK or spaCy, adjusting for legal jargon.
- Lemmatization: Reduce words to their base form (e.g., "running" → "run") using spaCy's lemmatizer.
- Domain-Specific Dictionary: Build or find a legal domain-specific dictionary (e.g., specialized legal terms or synonyms).

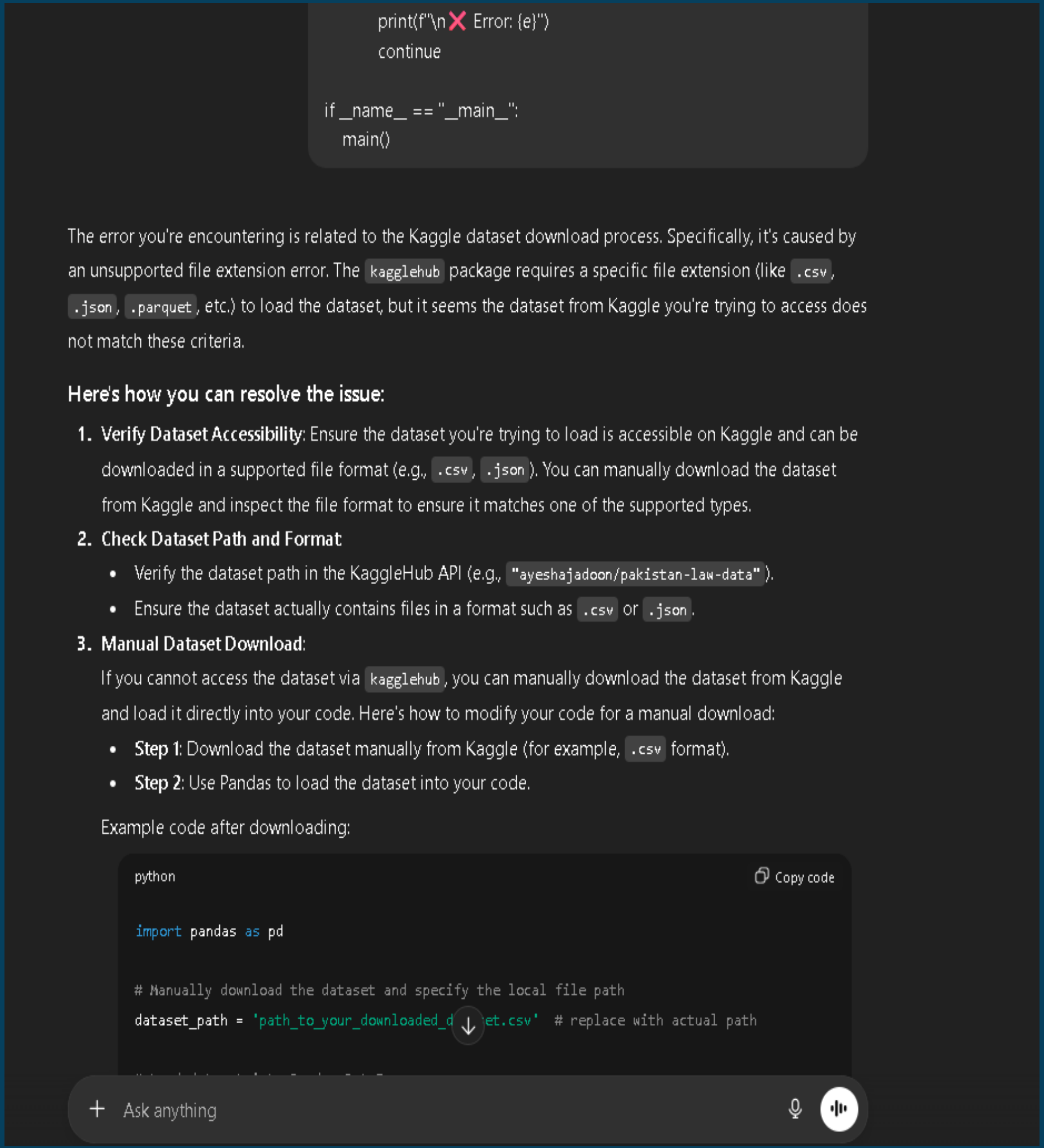
3. Indexing

- BM25/TF-IDF: Use `Scikit-learn`'s `TfidfVectorizer` or `Rank-BM25` (available through libraries like `Whoosh`) for baseline indexing. These methods calculate a vector representation of each document based on term frequency and inverse document frequency.
- Sentence Embeddings: Use pre-trained models such as `Legal-BERT`, `BERT`, or `Word2Vec` trained on legal corpora to generate sentence embeddings for documents. The `Hugging Face Transformers` library can be used to load these models.

```python
from transformers import AutoTokenizer, AutoModel
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
model = AutoModel.from_pretrained("bert-base-uncased")

def encode(text):
    inputs = tokenizer(text, return_te   s="pt")
    outputs = model(**inputs)
```

References & Attribution



```
        print(f"\n❌ Error: {e}")
        continue

if _name_ == "_main_":
    main()
```

The error you're encountering is related to the Kaggle dataset download process. Specifically, it's caused by an unsupported file extension error. The `kagglehub` package requires a specific file extension (like `.csv`, `.json`, `.parquet`, etc.) to load the dataset, but it seems the dataset from Kaggle you're trying to access does not match these criteria.

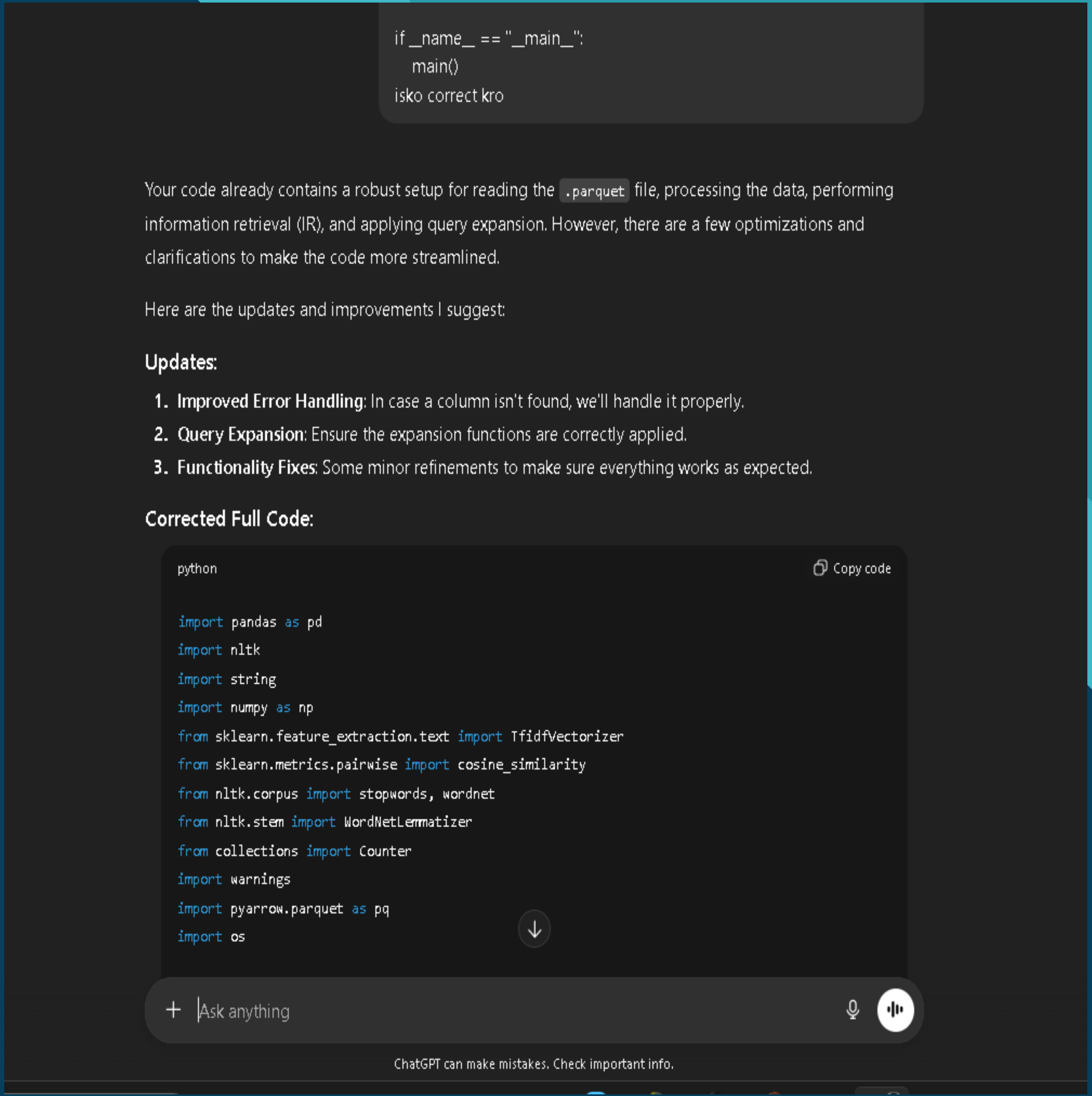
Here's how you can resolve the issue:

1. **Verify Dataset Accessibility**: Ensure the dataset you're trying to load is accessible on Kaggle and can be downloaded in a supported file format (e.g., `.csv`, `.json`). You can manually download the dataset from Kaggle and inspect the file format to ensure it matches one of the supported types.
2. **Check Dataset Path and Format**
   - Verify the dataset path in the KaggleHub API (e.g., `"ayeshajadoon/pakistan-law-data"`).
   - Ensure the dataset actually contains files in a format such as `.csv` or `.json`.
3. **Manual Dataset Download**:
   If you cannot access the dataset via `kagglehub`, you can manually download the dataset from Kaggle and load it directly into your code. Here's how to modify your code for a manual download:
   - **Step 1**: Download the dataset manually from Kaggle (for example, `.csv` format).
   - **Step 2**: Use Pandas to load the dataset into your code.

   Example code after downloading:

```python
import pandas as pd

# Manually download the dataset and specify the local file path
dataset_path = 'path_to_your_downloaded_d[illegible]et.csv'  # replace with actual path
```



```
if _name_ == "_main_":
    main()
isko correct kro
```

Your code already contains a robust setup for reading the `.parquet` file, processing the data, performing information retrieval (IR), and applying query expansion. However, there are a few optimizations and clarifications to make the code more streamlined.

Here are the updates and improvements I suggest:

Updates:

1. **Improved Error Handling**: In case a column isn't found, we'll handle it properly.
2. **Query Expansion**: Ensure the expansion functions are correctly applied.
3. **Functionality Fixes**: Some minor refinements to make sure everything works as expected.

Corrected Full Code:

```python
import pandas as pd
import nltk
import string
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from nltk.corpus import stopwords, wordnet
from nltk.stem import WordNetLemmatizer
from collections import Counter
import warnings
import pyarrow.parquet as pq
import os
```

Any
Questions?

Thank You