How csv file should be parsed into the RAG architecture

Before Proceeding with the information i will be assuming that the csv files are present with both **numeric values** and few **long characters** based on my research if we parse all the columns indiscriminately into the RAG Model we can face issues like:

1) **Text columns** are not being properly handled like if there are long characters present in the CSV file if it's not processed clearly the retrieval system might miss key information.

2) **Data preprocessing** should be included as well for feature extraction purposes that are likely to impact the retrieval side.

3) Text columns should be handled differently in context with **semantic search** and store numeric columns separately just like f1-score as you mentioned.

After a full examination and a thorough understanding the structure of fixed.csv & a1961 -43.csv (renamed it to SECTIONS.csv) I got to Know:

SECTIONS.csv:
Columns:
Name of the Statute, Section Number, Section Title, Section Text - THESE CONSISTS OF **INCOME-TAX ACT 1961, SECTION 1, SHORT TITLE, EXTENT AND COMMENCEMENTS AND A LARGE TEXT OF THE SECTION.**

FIXED.csv
Columns:
Document_id, case_title, judges name, date of judgment, issues, decision, cited cases, all_text - THESE CONSISTS OF **unique id, title of legal case, name of the judges, the date of judgment to be issued, list of legal issues, decision code, cited cases, and again a Large Text of the case.**

According my to Research on how csv file should be parsed into the RAG Model without any indiscrimination between columns and providing us the best output would be:

1) We need to perform **Data preprocessing** a traditional practice by reading the CSV data and to understand the characteristics of each column like what is the **data type**, to fetch **missing values** (if not there still a best practice) we can also extract each column and then convert it into **tolist()**, to find **inconsistencies** we also need to clean the data by removing **augmented data** and to apply **normalization** like **min-max scaling** in this case we can also use , **standardization** and **feature extraction** too.

2) Now if text data and numeric data are present then its a must to convert text columns into **embeddings** or **vectors** using **NLP** models like **BERT** and **OPEN AI** and etc. this helps for better **semantic searching** and **retrieval** of text based info for handling numeric data normalizing numeric values is also a must because for **filtering** or **scoring** to find context specific research and if dates are present then it's a good practice to convert it into datetime object.

3) For csv files dealing with **tabular data** doesn't work well in **vector data stores** they are basically designed to retrieve texts out of it, the methods we can try is by saving the whole CSV file into the vector store with a **text description** then after lookup feed the whole table into the prompt we need to look into **TAPAS** which is an open source project from **Microsoft** designed for tabular data the last step would be to save the data into a database then write an agent to query it a good way is to convert to sql db and then query a best way is to use knowledge graph if some of the cells are **text heavy**.

4) for Indexing methods too we can use **faiss** as well to retrieve relevant documents based on text data i will also source the necessary **links** as well and **youtube videos**.

5) These contain **heavy text** so to make it manageable, **chunk** the text into smaller **sections** based on **sections**, **sub-sections** or **paragraphs**. It's also important to ensure each chunk is meaningful and self contained to allow mechanism to bring relevant information.

6) For such Long Texts store structured data like **section numbers**, **case titles** and **judgment dates** in a relational database like **postgresSQL** to allow exact match queries based on structured data in context to CSV Files itself

7) It is important to limit the model's scope to the provided context itself using **prompt engineering** that guides the model to stay within the retrieved text and not to hallucinate and provide unnecessary or explicit Answers...using RAG Libraries like Haystack, Faiss and Weaviate.

Sources Referred to:

1) https://vivasai01.medium.com/overcoming-the-hurdles-navigating-rag-failure-when-handling-large-csv-files-19c65e7f60c0
2) https://aswin19031997.medium.com/enhancing-conversational-ai-with-retrieval-augmented-generation-rag-leveraging-csv-integration-3000322819eb
3) https://blog.gopenai.com/chat-with-large-csv-data-using-qdrant-langchain-and-openai-0d51ea06520a
4) https://arxiv.org/pdf/2307.03172
5) https://www.reddit.com/r/LangChain/comments/1dmj7p7/im_not_sure_i_understand_how_to_perform_rag_on/?rdt=40978
6) https://github.com/microsoft/AI-Chat-App-Hack/discussions/84
7) https://youtu.be/L1o1VPVfbb0
8) https://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html

- Ibrahim Sultan (3)