Author: Saurabh Arun Yadgire
Repository Link: https://github.com/yadgire7/csv-question-answering
Input: CSV file
Task: Create an embedding model to query a CSV file in natural language and return the output.
Output: {“column_name”: <EXTRACTED COLUMN NAME>, “value”: <MENTIONED/ EXTRACTED VALUE>, “row_ids”: [ROW IDs SATISFYING THE QUERY]}
I researched and experimented with various models and approaches to solve this problem. I have explained below all the approaches in detail. All the approaches are implemented using LangChain.

This approach can be scaled and modified for other problem statements with required data preprocessing and prompt engineering.

LLM used: Llama-2-13b-chat-hf
Embedding model used: sentence-transformers/all-MiniLM-L6-v2

Approach 1: Create a context from the embeddings of the CSV file

- Created a vector store using FAISS and used it as a retriever.
- Used similarity score metric to get k most similar documents(rows) to the query.
- This approach is infeasible as using a large k value increases the token length and leads to hallucinations and irrelevant results.

Approach 2: Self Querying

- Used Self Querying by providing metadata about the columns using “AtributeInfo”. For example: `AttributeInfo(`

```
            name="Product_Category",
            description="The category of the product ordered belongs to
        any of these or synonymous to these ['Apparel', 'Cosmetics &
        Personal Care', 'Groceries','Toys & Games', 'Electronics']",
            type="string",
        )
```

- The problem with this approach occurs when querying the date column. A solution to this problem is to convert the date to Unix epoch time during the preprocessing step.
- The above problem of scalability still persists as all the relevant documents cannot be retrieved with a small k value.

Approach 3:

- Design a custom CSV agent like create_csv_agent, create_pandas_dataframe_agent from LangChain that uses Python REPL and retriever (switches between these)
- This built-in agent is efficient but does not use embeddings exclusively for the CSV data.

Approach 4: Submitted solution

- Data Collection: read the CSV file using Pandas
- Preprocessing: Add row_id column, use 4 columns for embedding model and querying: Order_ID, Date, Product_Category, and Delivery_Distance
- Embedding Preprocessing: create column embeddings > find column name similar to query > check if column is categorical > if yes: create a formatted query > else pass the user query
- RAG: design a prompt to get pandas expression that returns the answer to the query > get row_ids that satisfy the query. Design a prompt to ge 'column_name' and 'value' from the Pandas expression
- Design output according to the format
- Return output

This approach can be scaled to multiple columns as the context length does not depend on the size of the data. We only use df.head() as the context for retrieval.

Other possible solutions: Create a custom agent using tools and functions from LangChain. A drawback of this approach is that as of now LangChain does not support open-source LLM functions.

References:

1. LangChain Documentation
2. HuggingFace Documentation
3. Blog on Question Answering Benchmark from LangChain
4. James Briggs' YouTube channel
5. Courses from DeepLearning.AI
6. StackOverflow
7. ChatGPT
8. Prompting Techniques
9. Langchain library code
10. Blog

Test Cases:

1. Query: product apparel
   Result:{'column_name': 'Product_Category',
   'value': 'Apparel',
   'row_ids': [0, 1, 2, 3, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 33, 34, 35, 36, 37, 38, 45, 46, 47, 48, 49, 50]}

2. Query: product machine
   Result: {'column_name': 'Product_Category', 'value': 'Electronics', 'row_ids': [89, 90, 91, 92, 93, 94, 95, 96]}

3. Query: product vegetables
   Result: {'column_name': 'Product_Category', 'value': 'Groceries', 'row_ids': [9, 10, 11, 12, 13, 28, 29, 30, 31, 32, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 82, 83, 84, 85, 86, 87, 88, 97, 98, 99]}

4. Query: orders between 1 july 2023 and 5 july 2023
   Result: {'column_name': 'Date', 'value': '2023-07-01', 'row_ids': [0, 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 14, 15, 16, 17, 28, 29, 30, 31, 33, 34, 35, 36, 39, 40, 41, 42, 45, 46, 47, 48, 51, 52, 53, 54, 58, 59, 60, 61, 65, 66, 67, 68, 82, 83, 84, 85, 89, 90, 91, 92, 97, 98, 99]}

5. Query: order id more than 4000
   Result: {'column_name': 'Order_ID', 'value': '4001', 'row_ids': [9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 45, 46, 47, 48, 49, 50]}

6. Query: delivery distance greater then 500 (with spelling mistake)
   Result: {'column_name': 'Delivery_distance', 'value': '> 500', 'row_ids': [14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 39, 40, 41, 42, 43, 44, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]}