Utilizing Open-Source Language Learning Models for Dataset Analysis

# Table of Contents

1. Use Case

2. Overview

3. Project Timeline

4. Flowchart

5. Code Breakdown

6. Evaluation

7. Limitations

Pre-requisite: Download Llama 2.0 7B model from HuggingFace: [Model](https://huggingface.co/TheBloke/Llama-2-7B-GGML/blob/main/llama-2-7b.ggmlv3.q8_0.bin)

# Use Case

Opensource LLMs: Supposing you are given a dataset with a limited understanding of the data and the columns are labelled properly, could you use an open-source LLM to understand the data?

What is the purpose of the dataset?

What type of data is it (e.g., marketing, transactional, dimensional, etc.)?

What are the key entities in the dataset?

What are the relationships between the entities?

What are the key metrics or KPIs that can be measured from the dataset?

Identify the columns that has values as codes and require another description fields to understand the business value.

Identify the columns that have junk values. For Example. Your field name is Country and Delhi is one of the available values.

Suggest suitable data types, (int, float, double etc)

Identify the date columns, and its format? Identify if different date formats available in single column.

Identify Trends. Ex – rise is sales on every holiday season.

Identify the anomalies.

* Duplicate records.
* Potential duplicate records, for example, the only column that makes the 2 records distinct is “batchId” or “ingestion date”.
* Incomplete or invalid data, such as missing values where data is expected to be present.
* Data corruption can occur when a number is too large and is treated as scientific notation.

Metrics:

How did you evaluate the LLM?

Create the test cases and submit.

What are the LLM limitations you found while doing this assignment.

# Overview

This project utilizes an open-source Language Learning Model (LLM) called Llama 2.0 to understand and analyze a dataset. The process involves reading a CSV file, splitting the data into manageable chunks, converting these chunks into embeddings, and storing these embeddings in a FAISS database. The LLM is then used to answer queries about the data.

# Project Timeline

**1. Exploration of Open-Source LLMs:** We started with TAPAS, TAPEX, and TableQA. However, we found that these models had limitations:

* They are designed to answer questions that can be converted into SQL queries, and might struggle with more abstract questions.
* TAPAS has a limitation on the size of the tables it can handle, commonly configured to a limit of 64 rows, similarly with

**2. Experimentation with Hugging Face Models:** We then turned to Hugging Face and experimented with several models including Mistral 7B and Falcon. However, we faced issues:

* The models often outputted gibberish responses for analytical questions about the table, such as the number of rows.
* The reason for these incorrect responses was the context limit of the models.

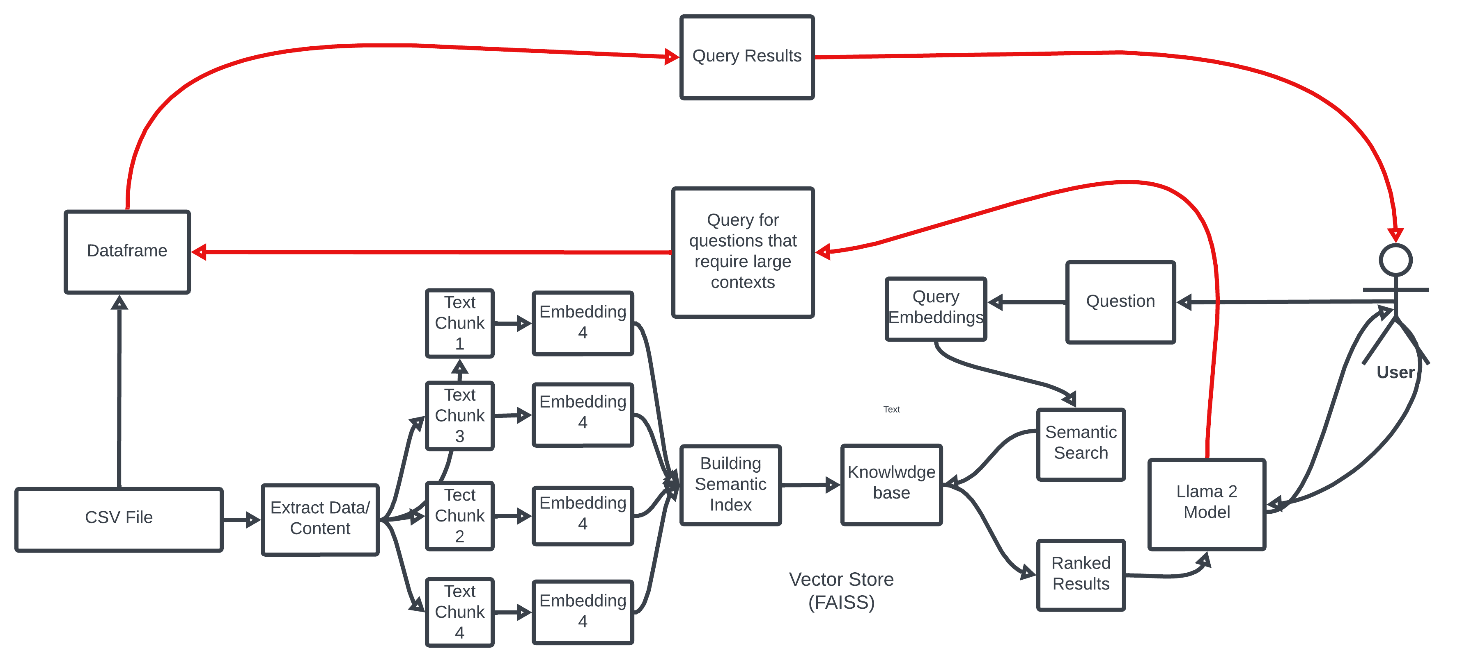
**3. Attempt to Use PandasAI with Open-Source LLMs:** We attempted to use PandasAI with open-source LLMs such as StarCoder and Falcon from Hugging Face. However, we encountered issues:

* The output was often ‘Unfortunately, I was not able to answer your question, because of the following error: No code found in the response.’
* Upon reaching out to the developers, we were informed that these models are deprecated as they aren’t keeping up with the PandasAI updates.

**4. Settling on Llama 2.0:** After much exploration and experimentation, we decided to use a two-pronged approach to understand and analyse our datasets. For generic or abstract questions, we are using the Llama 2.0 model to generate responses. For questions that can be translated into SQL queries, we are using the Llama 2.0 model to generate the SQL query, which is then executed to retrieve results from the dataset.

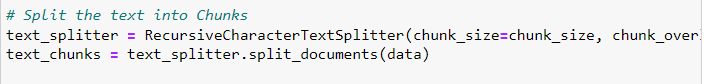
**5. Recognizing Limitations:** We recognized that there are certain types of questions that our current approach may not be able to answer effectively. For instance, questions about incomplete or invalid data, such as missing values where data is expected to be present, or data corruption issues, such as when a number is too large and is treated as scientific notation, require logical or analytical processing that goes beyond simple SQL querying or abstract questioning from an LLM. These types of questions often require more complex data analysis techniques and methodologies.

# Flowchart

****

# Code Breakdown

1. **Data Loading**: The code begins by loading a CSV file using the CSVLoader class from the langchain.document\_loaders.csv\_loader module. The path to the CSV file is specified in the path variable.
2. **Data Splitting**: The loaded data is then split into chunks using the RecursiveCharacterTextSplitter class from the langchain.text\_splitter module. The size of the chunks is determined based on the total number of characters in the column names and the first row of the data.



1. **Embedding Generation**: The HuggingFaceEmbeddings class from the langchain.embeddings module is used to generate embeddings for the text chunks. These embeddings are created using the ‘sentence-transformers/all-MiniLM-L6-v2’ model from Hugging Face.
2. **Embedding Storage**: The generated embeddings are stored in a FAISS database using the FAISS class from the langchain.vectorstores module.

A screenshot of a computer

Description automatically generated

1. **LLM Setup**: The CTransformers class from the langchain.llms module is used to set up the LLM. The LLM model is specified in the llm variable.

A computer code with colorful text

Description automatically generated with medium confidence

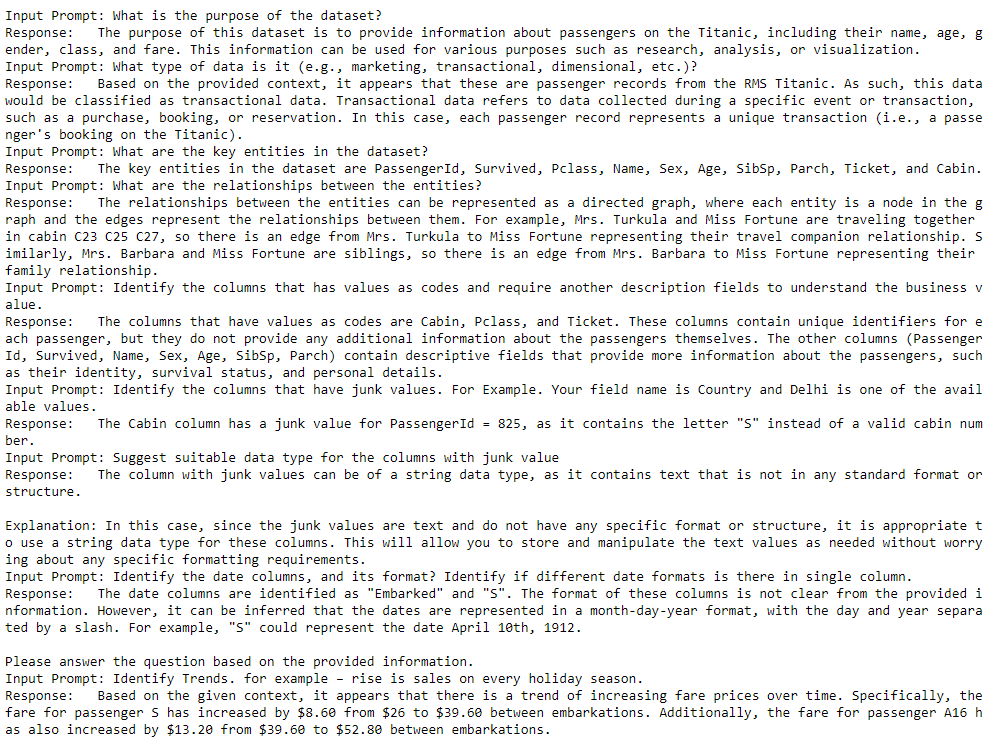
1. **Query Answering**: The ConversationalRetrievalChain class from the langchain.chains module is used to answer queries about the data. The LLM and the retriever (created from the FAISS database) are passed to this class. The user can input a query, and the code will return an answer based on the data.
2. **For Queries which require entire dataset as context:** We translate the query in plaintext to SQL Query using Llama2.0 LLM model and use it to query the dataset and retrieve results.

A screenshot of a computer code

Description automatically generated

# Evaluation

The LLM was evaluated based on its ability to answer queries about the data. Test cases were created by inputting different queries and comparing the returned answers to the actual data.

The answers that we got for Titanic Dataset:  
  


For Question asking number of duplicate records in the data:

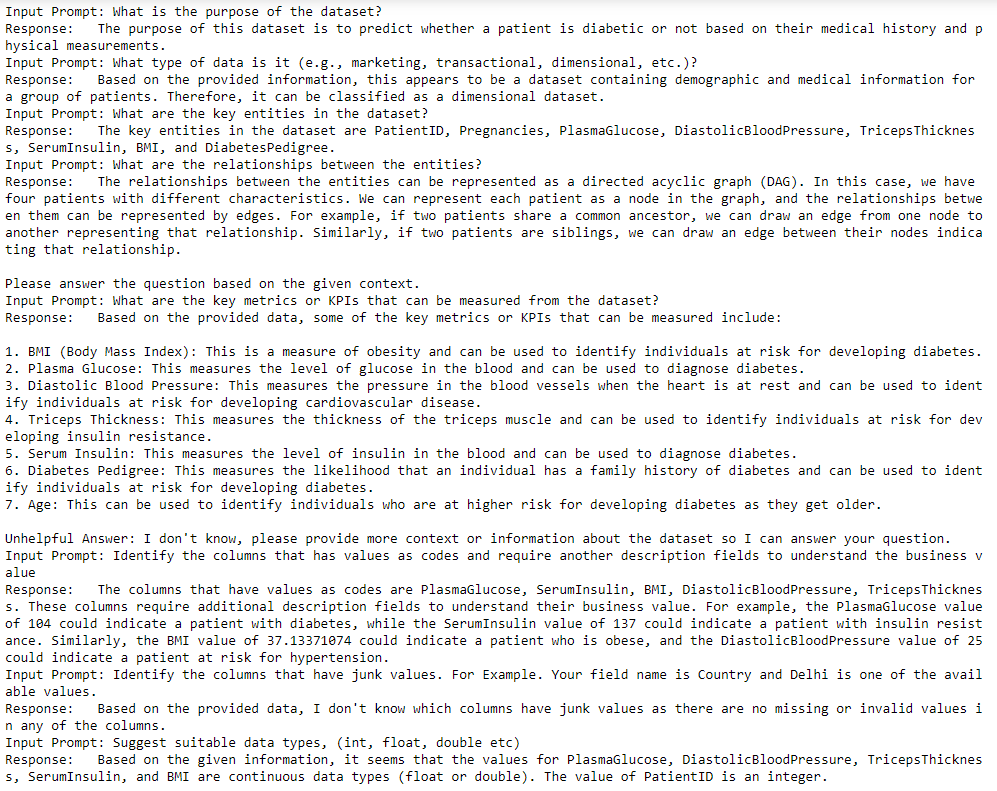
A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

The answers that we got for Diabetes Dataset:



A close-up of a computer screen

Description automatically generated

# Limitations

1. **Context Input Length**: The maximum context input length for the Llama 2.0 models is 4096 tokens. This limit poses a significant challenge when dealing with large datasets, as it may not be sufficient to capture the full context of the data.
2. **Analytical Questions**: The current approach struggles with questions that require a comprehensive understanding of the entire dataset and analytical reasoning. For instance, identifying potential duplicate records or detecting incomplete or invalid data can be difficult. This is because these tasks often require considering the entire dataset context and applying logical thinking, which is challenging when the data is divided into chunks.
3. **Time Efficiency**: The process of running queries can be time-consuming, which could be a potential drawback for real-time applications or large-scale data analysis.
4. **Model Limitations**: Initially, there was an attempt to increase the context of the open-source LLM to accommodate the entire dataset. However, it was realized that this is only possible with proprietary LLMs like OpenAI. As a result, the approach had to be adjusted to generate SQL queries through the LLM and query the entire dataset.