Chapter 1
CSV file analyzer using open source hugging face libraries

This chapter leverages open-source libraries and the Mistral-7B-Instruct large language model to analyze CSV files, performing tasks such as text summarization, semantic analysis, and similarity search. The workflow includes data loading with LangChain, summarization with Transformers, embedding generation with Sentence-Transformers, and similarity search with FAISS. The implementation also incorporates a query handling mechanism with `qa_chain` to

1.1 Introduction

In the modern era of big data, analyzing large datasets is crucial for deriving meaningful insights. This project focuses on developing a CSV file analyzer utilizing open-source libraries and a large language model (LLM) to perform various Natural Language Processing (NLP) tasks. The primary objectives include text summarization, sentiment analysis, and similarity search within the CSV data.

Objectives

The main objectives of this project are:

- To preprocess and analyze text data from CSV files.
- To leverage the capabilities of state-of-the-art language models for NLP tasks.
- To implement efficient data processing techniques for large datasets.
- To integrate multiple open-source libraries for a cohesive solution.
-

Tools and Libraries

The project utilizes a range of open-source libraries to achieve its goals:

- **Transformers**: Provides pre-trained models for various NLP tasks.
- **Accelerate**: Optimizes model training and inference performance.
- **Bitsandbytes**: Improves memory usage and computation speed for large models.
- **Huggingface_hub**: Manages and shares models and datasets.
- **LangChain**: Facilitates the building of complex text processing pipelines.
- **PyPDF**: Handles PDF manipulation tasks (although not directly used in this CSV analysis project).
- **Sentence-Transformers**: Computes sentence embeddings for clustering and semantic search.
- **FAISS**: Efficiently handles similarity search and clustering of dense vectors.

1.2 Methodology

The methodology involves the following steps:

Data Loading and Preprocessing:

Load CSV data using LangChain's CSVLoader.

Preprocess the data to clean and prepare it for analysis.

Text Analysis Using LLM:

Use the Mistral-7B-Instruct model to perform text summarization.

Implement text embeddings using Sentence-Transformers for semantic analysis.

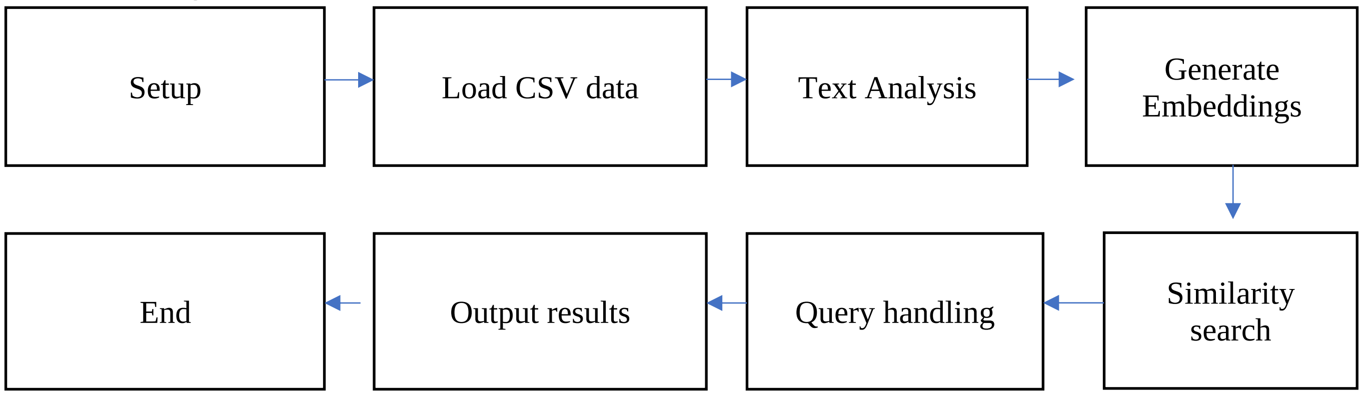
Similarity Search:

Generate embeddings for text data.

Create a FAISS index for efficient similarity search.

Perform searches to find similar texts based on a query.

4.2.1 Flow diagram



1.3 Implementation

1.3.1 Setup

```
!pip install -q -U transformers
!pip install -q -U accelerate
!pip install -q -U bitsandbytes
!pip install -q -U huggingface_hub
!pip install -q -U langchain
!pip install -q -U pypdf
!pip install -q -U sentence-transformers
!pip install -q -U faiss-gpu
!pip install -q -U chromadb
```

1.3.2 Loading CSV Data

```
from langchain_community.document_loaders.csv_loader import CSVLoader
```

```
loader = CSVLoader(file_path='rwa1.1.csv')
data = loader.load()
text_splitter = CharacterTextSplitter(chunk_size=1000, chunk_overlap=0)
chunked_docs  = text_splitter.split_documents(data)
embeddings = HuggingFaceEmbeddings()
```

1.3.3 **Running the LLM with Pipelines**

```
text_generation_pipeline = transformers.pipeline(
    model=model,
    tokenizer=tokenizer,
    task="text-generation",
    eos_token_id=tokenizer.eos_token_id,
    pad_token_id=tokenizer.eos_token_id,
    repetition_penalty=1.1,
    return_full_text=False,
    max_new_tokens=300,
    temperature = 0.3,
    do_sample=True,
)
mistral_llm = HuggingFacePipeline(pipeline=text_generation_pipeline)
```

1.3.4 Generating Embeddings and Similarity Search

```
from langchain.vectorstores import FAISS
db = FAISS.from_documents(chunked_docs,
                          HuggingFaceEmbeddings(model_name='sentence-
transformers/all-mpnet-base-v2'))

retriever = db.as_retriever(
    search_type="similarity",
    search_kwargs={'k': 4}
)
import sys
chat_history = []
query = input('Prompt: ')
result = qa_chain({'question': query, 'chat_history': chat_history})
print('Answer: ' + result['answer'] + '\n')
chat_history.append((query, result['answer']))
```

1.3.5 **Query Handling and Chat History**

```
import sys
chat_history = []
query = input('Prompt: ')
result = qa_chain({'question': query, 'chat_history': chat_history})
```

```
print('Answer: ' + result['answer'] + '\n')
chat_history.append((query, result['answer']))
```

1.4 Results

The implementation resulted in successful text summarization, embedding generation, similarity search within the CSV data, and effective query handling. Key outcomes include:

Summarization: Each text entry was effectively summarized using the Mistral-7B-Instruct model.

Embeddings: High-quality text embeddings were generated, facilitating semantic analysis.

Similarity Search: FAISS indexing enabled efficient and accurate similarity searches, identifying closely related text entries based on the query.

Query Handling: The **qa_chain** efficiently processed user queries and maintained chat history for context-aware responses.

Future Work

Potential extensions of this project include:

Enhanced NLP Tasks: Implementing more advanced tasks like named entity recognition, topic modeling, and custom text generation.

Performance Optimization: Utilizing distributed computing and further optimization techniques to handle even larger datase