Querying CSVs and Plot Graphs with LLMs

A project by Sheraphine Shovan M

Overview

This project creates a web-based application using Gradio to enable users to upload a CSV file, ask questions about the data, perform statistical analysis, and generate visualizations. The app leverages the LLaMA-2 model for natural language processing and the Sentence Transformers model for embedding text. Additionally, the application provides functionalities to analyze data, generate plots, and retrieve information using a ConversationalRetrievalChain

Code Explanation

Import Libraries

```
import os
import gradio as gr
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_community.llms import CTransformers
from langchain.chains import ConversationalRetrievalChain
from langchain_community.vectorstores import FAISS
from langchain.text_splitter import CharacterTextSplitter
from io import StringIO
```

Here we import the libraries after we install the necessary libraries.

Model Loading

Sentence Transformer model

```
# Load the sentence transformer model
sentence_transformer_model = "sentence-transformers/all-MiniLM-L6-v2"
sentence_transformer = HuggingFaceEmbeddings(model_name=sentence_transformer_model, model_kwargs={'device': 'cpu'})
```

This loads the Sentence Transformers model for text embeddings.

LLaMA2 Model

```
# Load the LLaMA-2 model
llm = CTransformers(model="D:\CSVAPP\model\llama-2-7b-chat.ggmlv3.q8_0.bin", model_type="llama", max_new_tokens=512, temperature=0.5)
```

This loads the LLaMA-2 model for natural language processing tasks.

Statistical Analysis

```
# Define the user-defined functions for statistical analysis and plot generation
def analyze_data(df, question):
    """
    Performs statistical analysis based on the user's question.

    This is a simplified example and will need more robust logic for real-world use.
    """
    response = ""
    if "mean" in question.lower():
        for column in df.columns:
            response += f"Mean of {column}: {df[column].mean()}\n"
    elif "standard deviation" in question.lower():
        for column in df.columns:
            response += f"Standard Deviation of {column}: {df[column].std()}\n"
    elif "correlation" in question.lower():
        correlation = df.corr()
        response += "Correlation Matrix:\n" + str(correlation)
    else:
        response = "I'm not sure how to perform that statistical analysis. Please ask a different question."
    return response
```

This function performs basic statistical analyses (mean, standard deviation, correlation) based on user queries.

Plot Generation

```python
def generate_plots(df, question):
    """
    Generates a plot based on the user's question.

    This is a s                                                    for real-world use.
    """
    plt.figure(figsize=(10, 6))
    if "histogram" in question.lower():
        column_name = question.lower().split()[-1].strip('"')
        if column_name in df.columns:
            plt.hist(df[column_name], bins=10, edgecolor="black")
            plt.title(f"Histogram of {column_name}")
            plt.xlabel(column_name)
            plt.ylabel("Frequency")
        else:
            plt.title("Please specify a valid column for the histogram.")
    elif "scatter" in question.lower():
        column_names = question.lower().split()[-2:]
        if all(column in df.columns for column in column_names):
            sns.scatterplot(x=df[column_names[0]], y=df[column_names[1]])
            plt.title(f"Scatter Plot of {column_names[0]} vs {column_names[1]}")
            plt.xlabel(column_names[0])
            plt.ylabel(column_names[1])
        else:
            plt.title("Please specify two valid columns for the scatter plot.")
```

(parameter) figsize: tuple[float, float] | None

```python
    elif "line" in question.lower():
        column_name = question.lower().split()[-1].strip('"')
        if column_name in df.columns:
            plt.plot(df[column_name])
            plt.title(f"Line Plot of {column_name}")
            plt.xlabel("Index")
            plt.ylabel(column_name)
        else:
            plt.title("Please specify a valid column for the line plot.")
    else:
        plt.title("I'm not sure how to generate that plot. Please ask a different question.")
    return plt
```

This function generates different types of plots (histogram, scatter plot, line plot) based on user queries.

Gradio Interface

```
# Define the Gradio interface
def main():
    with gr.Blocks() as demo:
        # Input components
        csv_file = gr.File(label="Upload CSV File")
        question = gr.Textbox(label="Ask LLaMA-2 about the data

        # Output components
        stats_output = gr.Textbox(label="Statistical Analysis")
        plot_output = gr.Plot(label="Plot")
        llama_output = gr.Textbox(label="LLaMA-2 Response")

        # Function to handle the upload and analysis
        def process_data(file, question):
            # Correctly read CSV into a DataFrame
            df = pd.read_csv(StringIO(file))

            # Perform statistical analysis
            stats = analyze_data(df, question)

            # Generate plot based on the question
            plot = generate_plots(df, question)
```

```
            # Create vector store for the CSV data (for LLaMA-2)
            text_splitter = CharacterTextSplitter(chunk_size=1000, chunk_overlap=0)
            texts = text_splitter.split_text(df.to_string())
            vectorstore = FAISS.from_texts(texts, sentence_transformer)

            # Create the ConversationalRetrievalChain
            chain = ConversationalRetrievalChain.from_llm(llm=llm, retriever=vectorstore.as_retriever())
            llama_response = chain.invoke({"question": question, "chat_history": []})["answer"]

            return stats, plot, llama_response

        # Connect components and function
        gr.Button("Analyze").click(fn=process_data, inputs=[csv_file, question], outputs=[stats_output, plot_output, llama_output])

        # Run the interface
        demo.launch(share=True)

if __name__ == "__main__":
    main()
```

This sets up the Gradio interface, which includes:

- File upload for the CSV file
- Textbox for user queries
- Output components for displaying statistical analysis, plots, and LLaMA-2 responses
- The process_data function handles data reading, analysis, and generating responses using the ConversationalRetrievalChain.

Conclusion

This project provides a comprehensive tool for analyzing CSV data and querying insights using advanced NLP models. It demonstrates integration of various Python libraries and models to create a user-friendly interface for data analysis and visualization. Further enhancements could include more sophisticated natural language understanding and additional types of statistical analysis and plots.

Important Links

Project Github: https://github.com/sheraphineshovan/LLaMA2-CCVapp

My Resume: https://drive.google.com/file/d/16X8rP-FBkvDobvb4JinmW5z_haAI1x8o/view?usp=drivesdk

My LinkedIn: https://www.linkedin.com/in/sheraphine-shovan-m/