**DS2002 Data Project 2**  
**Group Members: Shaina Banduri, Maya Shah, Diya Gupta**

For Data Project 2, our group developed a Flask-based chatbot assistant that responds to user queries about movies and shows using two data sources: a cleaned Netflix dataset and live data from The Movie Database (TMDB) API. This project combined data engineering, API integration, backend development, and cloud deployment. The chatbot was designed to search the local dataset first and fall back on the TMDB API if needed, ensuring accurate and comprehensive responses. It was deployed on Google Cloud Platform (GCP) and is publicly accessible.

**Project Focus and Data Selection**

We chose streaming content as our topic due to its relevance and the abundance of available metadata. For our local dataset, we used the Netflix Titles Dataset, which includes information on thousands of titles. As our live API source, we selected the TMDB API, which offers detailed movie and TV show data such as overviews, ratings, and release dates. This pairing allowed us to cover a wide range of queries, from Netflix-specific titles to more general film and television content.

**ETL Process**

The ETL (Extract, Transform, Load) script was written in Python using pandas. It reads the raw Netflix CSV, removes unnecessary columns like show\_id and description, drops rows with missing values in critical fields, and standardizes date formats. We saved the cleaned data as cleaned\_netflix.csv, which the Flask app loads on startup.

We ensured the dataset only included complete, well-formatted entries to optimize search performance and improve the chatbot’s reliability. Decisions about which columns to keep and how to transform the data were made collaboratively, with attention to both performance and usability.

**Flask Chatbot Design**

The Flask application exposes a single /chat route that accepts POST requests. When a user sends a query, the app searches the local Netflix dataset for a matching title. If a match is found, it returns the title, type (Movie or TV Show), and release year. If not, it calls the TMDB API to fetch relevant data and returns additional details like the overview and average rating.

We used helper functions to isolate the API call logic and ensure that the main application code remained clean and readable. We also implemented error handling to return clear messages when no match is found or if the API is temporarily unavailable.

**Group Collaboration**

Our team divided the workload to leverage everyone’s strengths. One member handled the ETL pipeline and Netflix dataset processing, another implemented the TMDB API integration, while others worked on the Flask routing, error handling, and deployment. We used GitHub for version control and coordinated updates to avoid conflicts. Each group member was responsible for testing their components and reviewing pull requests, allowing us to maintain high code quality and ensure alignment across tasks.

**Deployment on GCP**

We deployed the completed chatbot using a GCP virtual machine running Ubuntu. After configuring the instance to allow HTTP traffic, we transferred the project files, installed all dependencies from requirements.txt, and launched the Flask app. The app runs on port 5000 and is accessible via the VM’s external IP address. We also confirmed that the chatbot responds to queries accurately and that both the local dataset and TMDB fallback logic function correctly.

**Challenges and Outcomes**

One of the main challenges was ensuring that user queries matched entries in the local dataset despite differences in capitalization or formatting. We addressed this by converting titles to lowercase during both storage and lookup. Another challenge involved coordinating the API integration so it would only be used when necessary. Clear function boundaries and modular code helped us resolve this efficiently.

This project allowed us to apply the core concepts of data ingestion, cleaning, and integration with web services. We also strengthened our understanding of backend development using Flask and how to deploy data-driven applications in a cloud environment. The ability to connect multiple data sources and build a responsive, publicly accessible assistant demonstrates the practical value of combining data engineering and software development skills.