BA882 Project Deliverable 2

Improvements from on Phase 1

1. From EtLT to ETL

In the first phase, we didn't focus extensively on the content of the columns provided by the data sources. Instead, most of our initial transformations were centered around formatting adjustments, such as converting dates from strings to datetime objects and extracting numerical video lengths from strings to facilitate future analysis. It was only during the visualization phase that we made temporary adjustments to the content of certain columns. This process represents our EtLT approach.

However, during this phase, we realized that one important adjustment couldn't remain temporary: the "title" column in the YouTube API data. Since the title of a show or movie serves as the unique key connecting multiple data sources, and the YouTube data's "title" column includes additional descriptors of the video content (such as "trailer" for a movie or "highlight" for a show), we decided to incorporate the removal of these descriptors into the transformation process in the write script. This ETL process ensures that the newly generated "extracted_title" column is consistently present.

2. Use Insert Logic for Data Upsertion

In the first phase, our loading logic involved replacing the stage tables with new daily data extracted each time, which in most cases included both historical and new records. However, we discovered that due to issues with the data sources, sometimes records included in the previous update were missing in the current one leading to loss of data. So during this phase, we modified our approach to use an insert logic: we performed a historical load and then subsequent updates to insert new incoming records everyday. With this approach, we are able to ensure the completeness of our data.

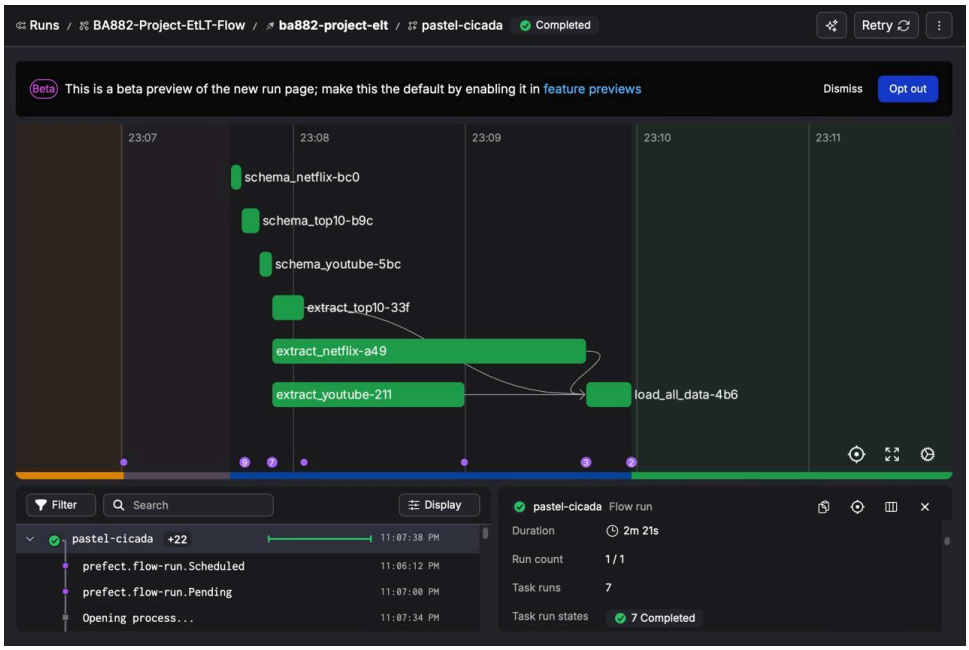
3. Fix Prefect Issue

During the second phase, we accidentally discovered that our data was not being updated. After checking all the cloud functions and confirming there were no errors, we suspected an issue with Prefect. Eventually, we found that some scheduled runs had failed in the Prefect interface.

By comparing the Professor's example file with the file we used for deployment, we realized that we had made an error in specifying the entrypoint: we had pointed the entrypoint to the deployment file, whereas, in reality, we were supposed to use the deployment file to instruct Prefect to trigger the entire flow using another python flow file that we have.

In addition to correcting the entrypoint to the appropriate file, we also organized all the files into a single folder. Originally, our "elt" file was in a separate folder called "flows",

while our deployment file was in a subfolder named “flows” under the “prefect” folder. To avoid potential location errors caused by the folder naming conflict, we moved both files into the “flows” folder under the “prefect” directory. Our new attempt was successful:



Developments in Phase 2

1. Dataset Expansion

For the purpose of building machine learning models using Netflix data from the last phase, we had to go back and extract additional data for the years 2020 and 2021 to increase the size of our dataset.

2. Machine Learning

We aimed to use unsupervised Machine Learning techniques to generate Netflix content recommendations based on the input provided to the model.

3. Recommendation Algorithm Implementation

We implemented a K-Nearest Neighbors (KNN) algorithm with cosine similarity for our recommendation system. This approach allowed us to:

- Calculate similarities between shows or movies based on vectorized text and normalized quantitative attributes
- Identify similar titles within each content category

4. Feature Selection and Engineering

We initially considered the following features for both movies and series:

- Cast
- Directors
- Genre
- Episode count (series-specific)
- Season count (series-specific)
- Overview
- Title

Given the structural differences between movies and TV series, we developed separate recommendation models for each category. This tailored approach better captured the nuances unique to movies and series, especially considering that our movie data lacked episode and season count information.

Another feature that we dealt with was “runtime”. Although we included “runtime” as a feature, during evaluation, we observed that:

- Runtime bias led to recommendations predominantly based on similar durations
- This overshadowed other crucial features

After careful consideration, we removed runtime from our feature set to:

- Allow for more balanced influence of other features
- Prioritize content similarity based on thematic and creative elements
- Provide users with more diverse and potentially unexpected recommendations

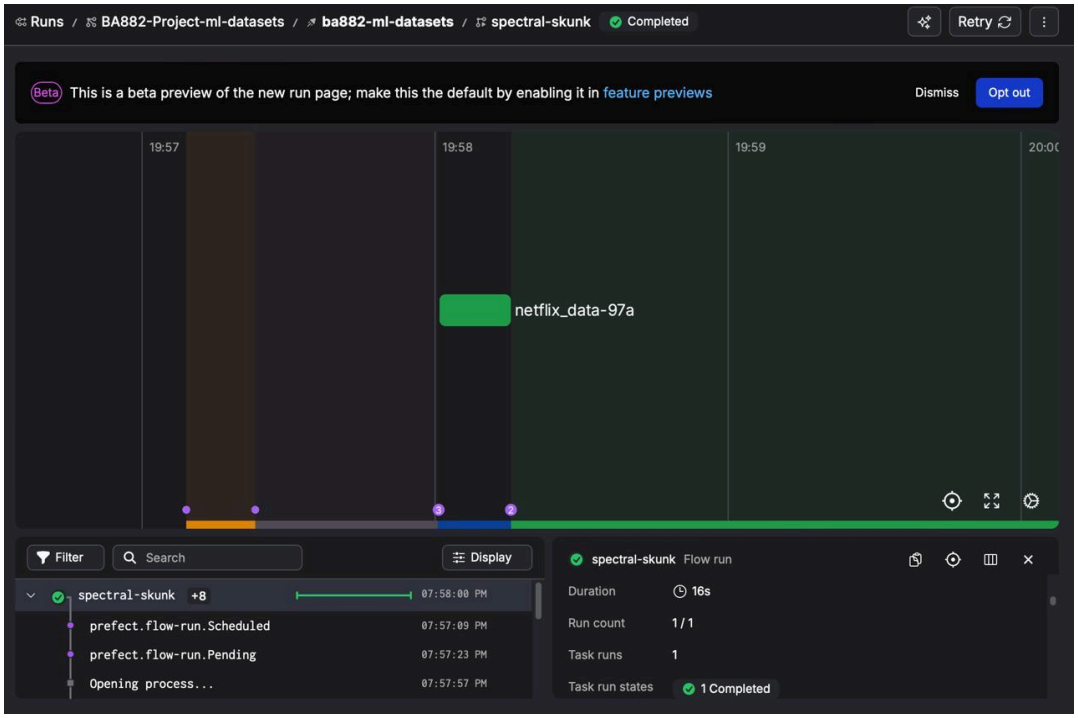
5. Pipeline

Since we have chosen to use serverless functions to deploy the machine learning component on the cloud, we need to download the final Netflix API data stored in MotherDuck to Google Cloud Storage. This process is integrated as an extension of our previous pipeline, and we continue using Prefect for scheduled deployments.

We have divided the model into two Cloud Run Functions: one for reading data, training the models, and storing the trained models, and the other for deploying the stored models. In the Cloud Run function’s testing interface, we can input information about one or more movies/shows and receive the model’s top 10 recommendations for similar content.

We further extended the pipeline to store trained model artifacts in our data warehouse on MotherDuck, creating a comprehensive log of daily model generations. This enhancement ensures that the ML process runs on a daily schedule, with each execution’s results saved for model comparison and historical tracking, allowing us to monitor performance trends and evaluate model improvements over time.

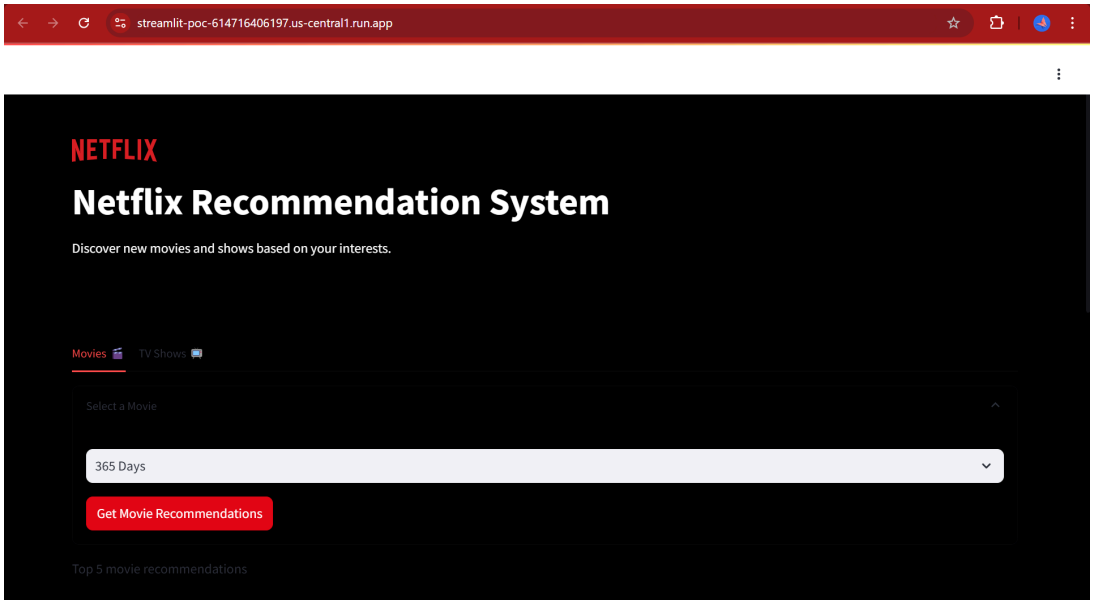
Below is a screenshot of the pipeline that we deployed on Prefect Cloud:



6. Streamlit UI

We began building a user interface in Streamlit to allow users to interact with the model. The goal was to let users select a movie or show from a list and, based on their choice, fetch the Top 5 recommendations using our deployed KNN model. While we made significant progress, we encountered some unresolved errors and couldn't fully develop the front-end. Below is a screenshot of our current progress, along with the URL to view the interface in its current state.

URL: https://streamlit-poc-614716406197.us-central1.run.app/



Limitations

- Despite expanding the dataset to include data from 2020 and 2021, it remains relatively small for machine learning applications. With around 3,500 rows, the system may not fully capture the diversity of titles or genres, potentially limiting recommendation accuracy. We will try to get more data for improving the recommendations. The model accuracy is also too high which implies a possibility of overfitting.
- The model does not incorporate user preferences, ratings, or viewing history, which are typically critical factors in recommendation systems. Recommendations are therefore generalized and may lack personalization.
- We only focussed on Netflix API for this task and did not use YouTube API or other tables. This limited our recommendations to some extent and also did not give us a holistic view of the entire dataset.

Future Scope

- We aim to finish developing on Streamlit successfully by the next phase in order to provide an interactive UI for users to interact with and get some cool recommendations based on their choice of movie/show.
- We also plan on doing a sentiment analysis on the overview and comments sections of the dataset to help us with recommendations. We will look into some existing models if we can find something related to our dataset. Else we will build a custom model that specifically caters our data. Furthermore we will compare it with LLMs to see what is performing better and inculcate that in the final model.