Project Report :

Milestone 3:

Monitoring and Continuous Deployment

Github Link :

https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML

Ameya Morbale
Ankit Kumar
Prakhar Pradeep
Praveen Ramesh
Kedi Xu

Containerization

Docker

The team used docker to containerize the entire flask application. This provides various benefits like isolation, portability, scalability, and versioning. Since our entire app is written in python 3.8 we used that as our base image and exposed port 8082.

LINK TO DOCKERFILE:
https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/Dockerfile

Switching between models

To switch between models we created our own load balancing system that would use a config.json file to select behavior of the loadbalancer. The configuration file provides an option for a default model. If the default model is none, then the config file goes to the next part. The next part has conditions and a model to run if that condition is true. Currently our system is configured to distribute the requests to model_v1 and model_v2 with a 1:1 split. These conditions can be changed and were changed when experimenting. The config.json file and model directory (where all models are stored) are checked for update after every 5 seconds, through an asynchronous function, this allows the flask app to update the load balancer's behavior without any downtime.

LINKS

Config.json
https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/Server/congif.json

Load balancer
https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/Server/load_balancer_helper.py

App
https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/Server/app.py

Automated model updates

To automatically update the models, we wrote a python script that takes one argument - time in minutes for data collection. The script runs the entire pipeline: collect data, process data, train a new model, evaluate (offline) the new model. The script also checks the data and model directory on the disk and determines the name of the dataset and model (data_X and model_vX).

Next we created a shell script called "run_pipeline.sh" and added the necessary code to call the complete_pipeline.py script with the required arguments - *python3 complete_pipeline.py 15.* This command executes “complete_pipeline.py” code and collects new data for training through kafka for 15 mins and trains a new model. The updated model and datasets are stored in respective folder structures

Next, we ensured that the shell script "run_pipeline.sh" had the appropriate permissions to be executed. To do this, we used the **chmod +x run_pipeline.sh** command, which makes the script executable ie. if we run **./run_pipeline.py** command in the same directory as the complete_pipeline.py file, the python file is triggered to execute which automatically updates the data and model.

After that, we tested the shell script manually by running it with **./run_pipeline.sh** to ensure that the complete_pipeline.py script was executed correctly and the entire pipeline (data collection, data processing, model training, and model evaluation) was executed as expected.

We have made the design decision to keep automatic model updates to be manual and human triggered to enable autonomy of updates. They do not exceed routine tasks as running one shell script - **./run_pipeline.sh.**

For deploying the latest model, again we chose to keep it manual to enable autonomy and experimentation. We just have to edit the model version string in a json file (model.v_x.pkl where x is 1,2,3…. in the a config.json file)

Link to complete_pipeline.py
https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/Pipeline/complete_pipeline.py

Link to run_pipeline.sh
https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/Pipeline/run_pipeline.sh#L2

Link to config.json
https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/Server/congif.json#L1-L13

Monitoring

The monitoring infrastructure has been established using Grafana and Timescale CloudDB. Our system's movie recommendations and user telemetry (movie watch history data) are streamed through Kafka and pushed to Timescale CloudDB using the PostgreSQL Python client. The data is stored in two separate tables: one for recommendations and one for user telemetry. This data is then transferred from TimescaleDB to Grafana for visualization purposes.The data storing code is executed using the 'nohup' command, which allows for continuous storage of real-time data

Two dashboards have been created for monitoring purposes.The first (a) dashboard monitors system availability by presenting the HTTP response status and service latency within a 1-minute timeframe. The second (b) dashboard focuses on online evaluation by monitoring the quality of recommendations, which is determined by the percentage of users who watch at least one of the twenty recommended movies within a one-hour timeframe. These dashboards are updated every 5 seconds.

Alerts are generated when the recommendation system's latency exceeds 400 ms, signaling potential issues with the system. These alerts are triggered when latency surpasses the 400ms threshold. Grafana is configured to send these alerts to the team's Slack channel using a webhook.

Pointers to code:

Code for storing movie recommendation log data from kafka stream to timescale cloudDB:
https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/kafka_timescale.py

Code for storing user telemetry log data from kafka stream to timescale cloudDB:
https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/kafka_telemetry_timescale.py

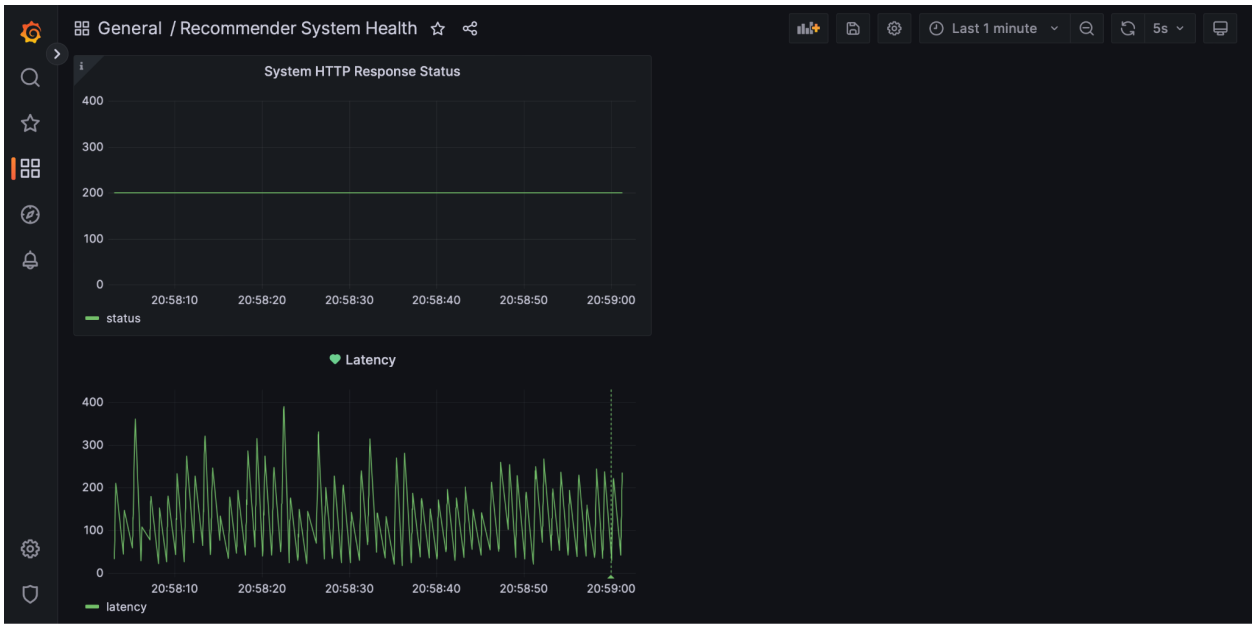
SQL query to display the metric in Grafana dashboard:
https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/grafana_query.sql

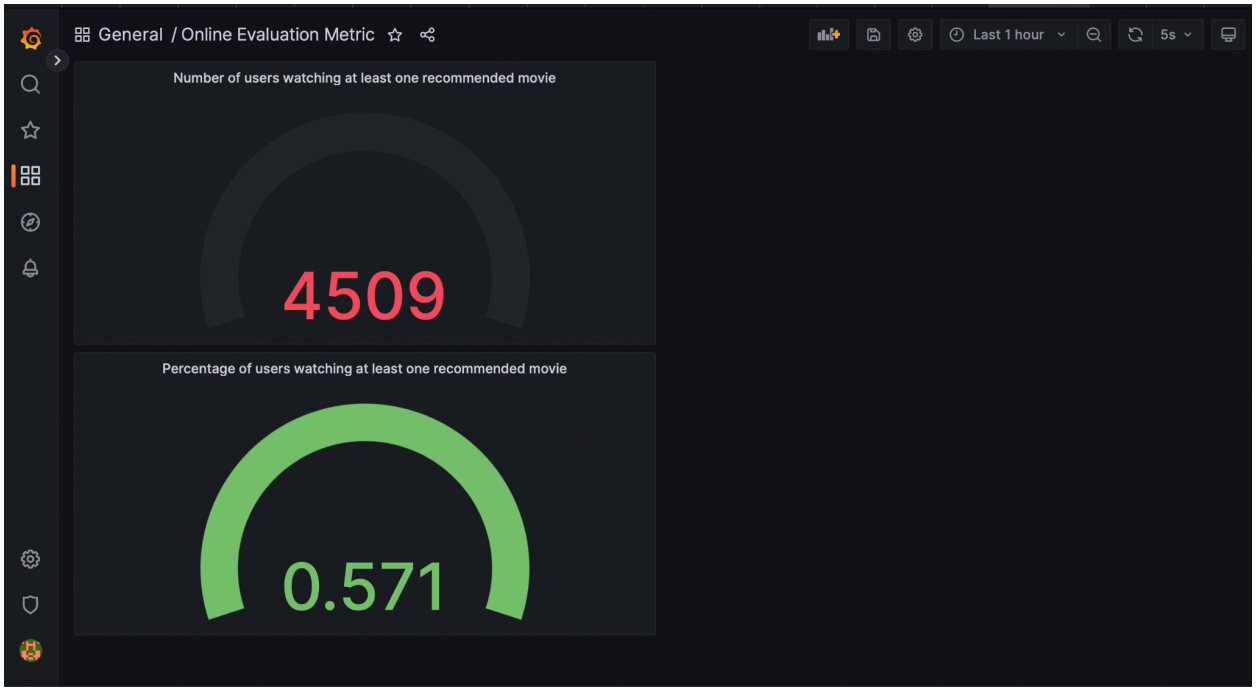
Links to Grafana Dashboards:
Credentials - Username - admin , Password - admin (Please skip the password update popup after logging in to see the dashboard)

a) System availability and latency:
http://128.2.205.109:3000/d/xbGAQHEVz/recommender-system-health?orgId=1&refresh=5s&from=now-1m&to=now

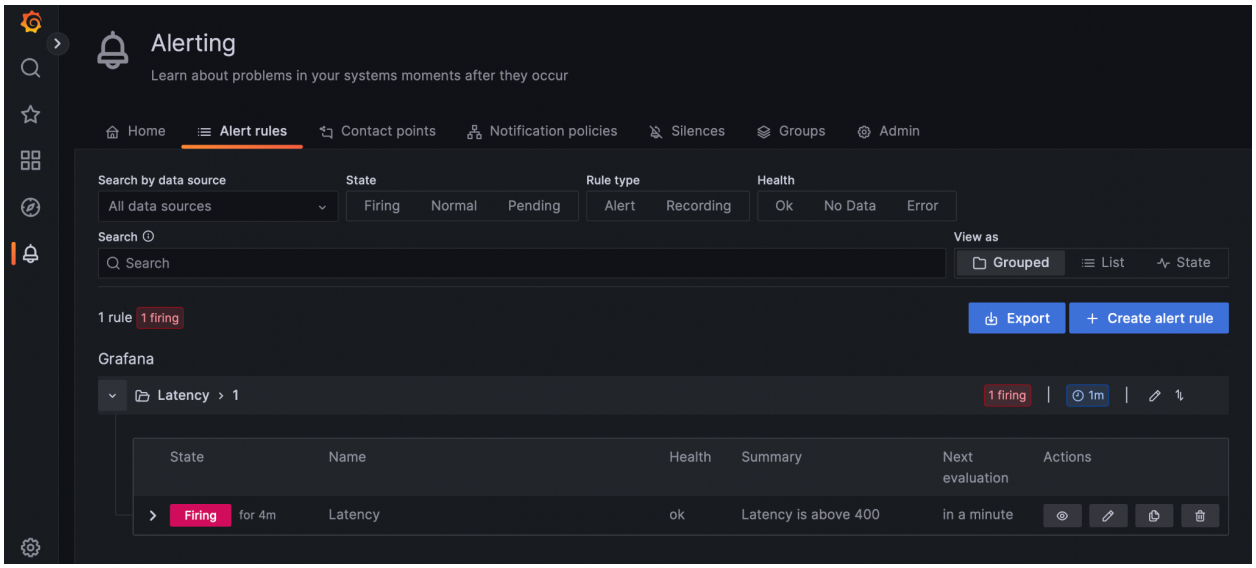
b) Online quality metric:
http://128.2.205.109:3000/d/hBVxMvP4z/online-evaluation-metric?orgId=1&refresh=5s&from=now-1h&to=now

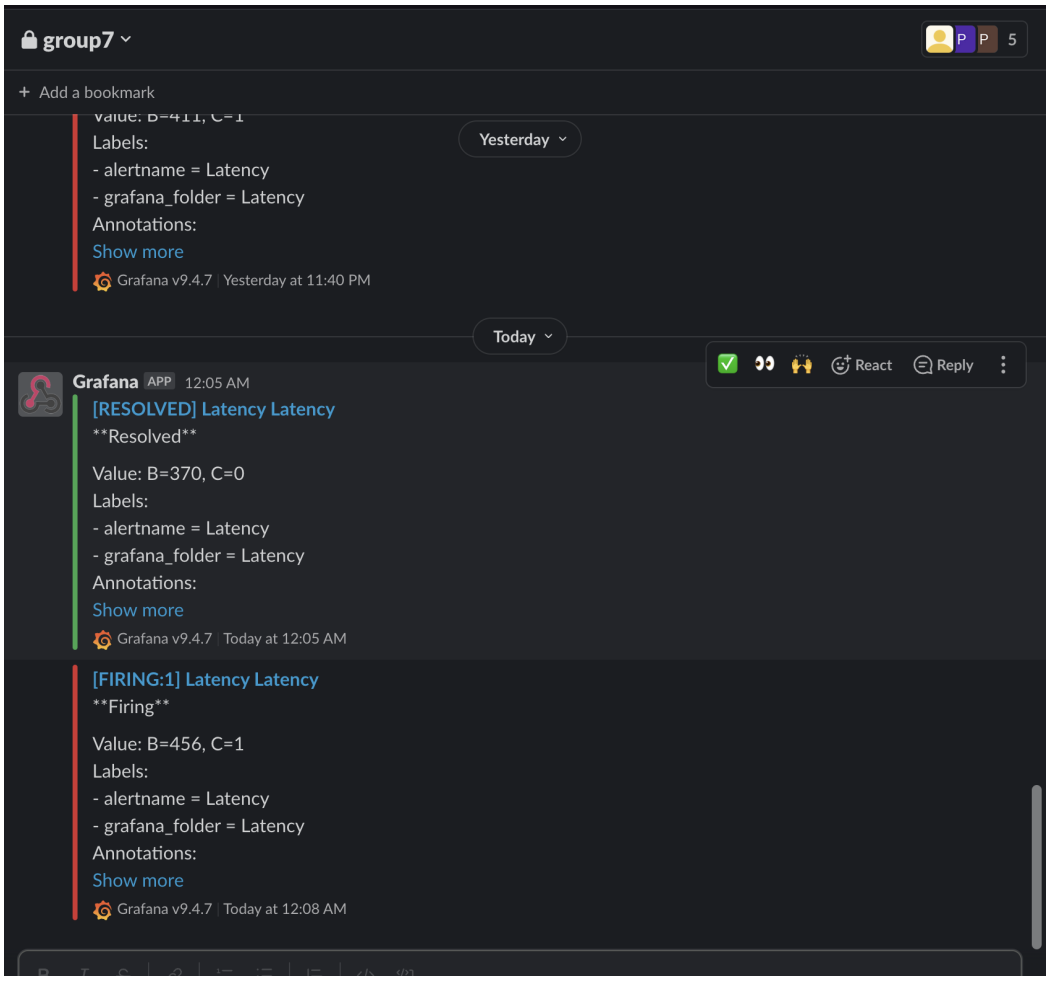
Below are the screenshots of the dashboards and alerts for reference.





Alert page and slack notification screenshot:





Experimentation

Overall, we wrote a pipeline of data processing, user splitting, model training, model serving and online evaluation for the experimentation infrastructure. When doing experiments, the user should first split the user data using https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/experimentation/data_split.py, then train models for experimentation https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/experimentation/models_training.py, then run the experiments online and use the experimentation models to serve users https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/experimentation/models_serving.py. And finally report the experiment results https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/tree/main/experimentation/OnlineEvaluation.

- User Splitting

You can split the users in two parts of the pipeline. First, when training the models, you can train a different model for different groups of users. In the user splitting part, you can write a plugin for the split_users function, so that you can split users by different criteria (for example, their demographic features).
https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/experimentation/data_split.py#L69
The default splitting function split the users by gender, https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/experimentation/data_split.py#L63. Note that we have collected a user details file to store user features, and join it with the users' moving rating data.
In the model serving part, different models are used to serve different user groups.
https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/experimentation/models_serving.py#L58.

- Quality Tracking

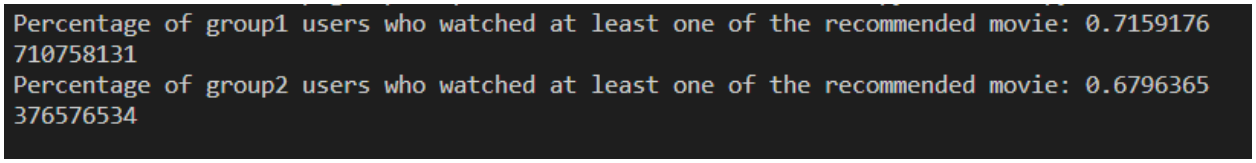
The experimentation infrastructure tracks the quality of recommendations through the average of Click-Through Rate (CTR). CTR is defined as the proportion of users who click on a recommended movie within one hour of the recommendation, indicating its relevance to the user's interests.
To implement this, first, we wrote a script to get recommendation predictions from the kafka stream.
https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/experimentation/OnlineEvaluation/get_Service_Predictions.py.
Then, we get user telemetry within one hour of the recommendation:
https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/experimentation/OnlineEvaluation/get_User_Telemetry.py.
Finally, we compute the average CTR
https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/experimentatio

n/OnlineEvaluation/get_evaluation_metric.py#L6. We define the recommended set as the top ten of our recommendation, and as long as the user clicks on one of them, it is counted as clicked.

- Results Reporting

After the metrics are calculated, they are directly printed in the command line.
https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/experimentation/OnlineEvaluation/get_evaluation_metric.py#L34

Screenshot of testing on two different models, users randomly split by id:



Provenance

Data's provenance, which includes details about its production, modification, and use, is referred to as its source, lineage, or history. Scientific research, data analytics, machine learning, and artificial intelligence are just a few of the domains where provenance is crucial. It aids in comprehending data reliability, repeatability, and quality.
A well-liked open-source program called **Data Version Control** (DVC) allows data scientists and machine learning professionals to manage and track changes to data, code, and models while also logging provenance information. We will look at how DVC may be used to manage provenance in data-driven projects in this report.

Provenance Tracking with DVC:
DVC provides several features that facilitate provenance tracking in data-driven projects:

1. **Data Versioning:** Data scientists can version their data files using DVC, which makes it easier to trace changes to data across time. Data files can be kept in a private file server, Amazon S3, Google Cloud Storage, or another remote storage system. Data scientists may precisely identify and recreate data used in a project thanks to DVC, which tracks each version of a data file using a distinct hash-based identification.
2. **Model Versioning:** Data scientists can track changes to models. Thanks to DVC's integration with well-known version control systems like Git. This makes it simple for team members to collaborate and aids in storing the dvc mapping on git and actual model pkl file on the server.
3. **Reproducibility:** Data scientists can create and manage data pipelines, which record the processes used to process data, using DVC. These pipelines can be versioned with the data and code, ensuring that the outcomes are entirely reproducible. DVC further provides the storing of interim findings, minimizing duplicate calculations and accelerating iterative processes.

Experiments in movie recommender system:

1. **DVC config file:** This config file will show the path of the remote repository. In our case, we have used the local repository to store the files.
   https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/.dvc/config

```
remote = data_repo
['remote "data_repo"']
    url =
/home/team07/Milestone3/group-project-s23-The-hangover-Part-ML/Data_repository
/
```

2. **Data versioning:** There are the different versions of data are used in our application. All of them are versioned using the DVC. The DVC files are present on the git repository. https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/tree/main/Data

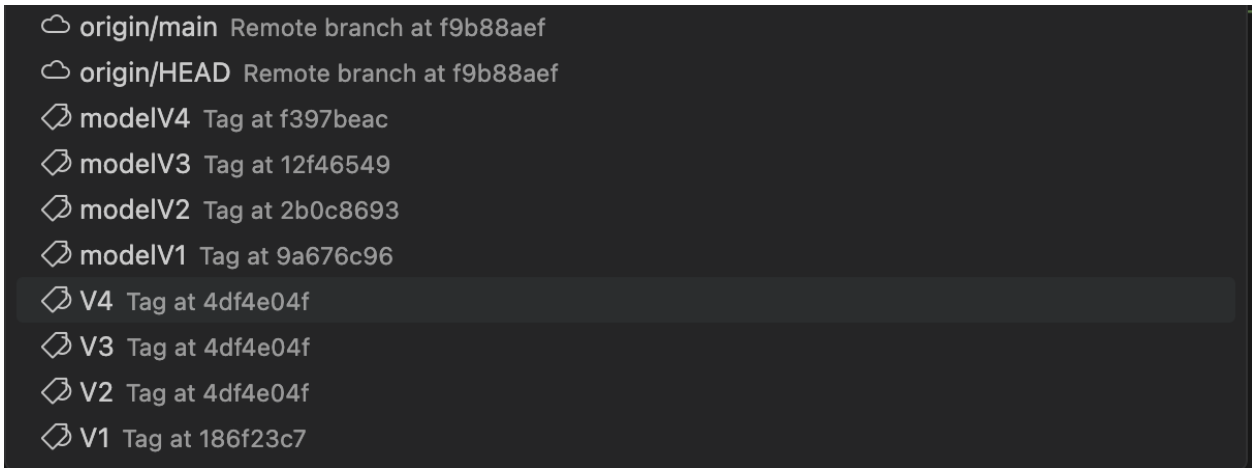
   These dvc files can be used to retrieve the old data files by the following command:
   dvc checkout Data/dataset_1.csv.dvc

   This command will pull the actual dataset from the repository handled by dvc. If we want to store any dataset in the dvc remote repository. Following commands needs to be executed:

```
#dvc add Data/dataset_2.csv
#git add Data/dataset_2.csv.dvc
#git commit -m “Dataset update”
#git push
#dvc push
```

   Commit hash has been attached to this dataset by which we can fetch this dataset anytime we want using:
   git checkout <Commit hash>

   Below are the commit hash generated in our case{ four datasets and four models}



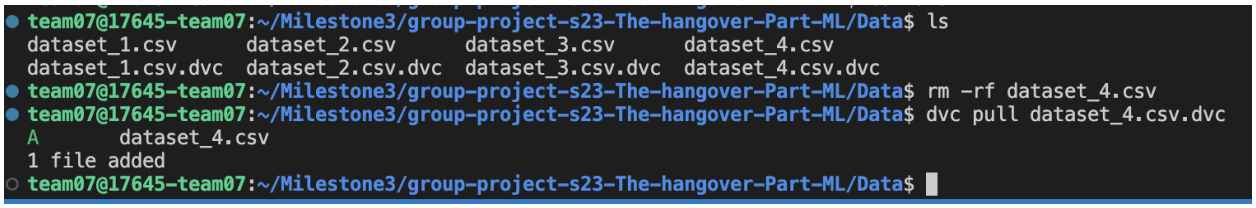
3. **Model versioning:** There are different versions of model are used in our application. All of them are versioned using the DVC. The DVC files are present on the git repository. https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/tree/main/Models

   We do not need to store the big pkl files on the git repository, but we can access them from the remote dvc repository whenever we want using commit hash.

4. **Tracking:** We can track using the pull request. Whenever we want any specific model and train them with specific dataset. We can use the pull request to fetch the data and the model from dvc remote repository.

   dvc pull <dvc file path>

   Actual versioning is done at the git because it maintains the dvc files and use the corresponding dvc to detch the data from the dvc repository.



   Similarly, we can do for any old version of data and model.

5. **Versioning Tracking:** We have collected logs using flask which shows which version is being used in prediction task. Our pipeline version is same from starting. The last column shows the model version and dataset version.
   https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/Server/app.py