MLOps Assignment Report

Krishna Chaitanya Gorle
Roll No: DA25M011

February 15, 2026

Abstract

This report is to discuss what challenges I have faced while manually versioning data and how I am managing training and deployment of the model

1 Challenges in Manual Data Versioning

Q1: What was the most difficult part of managing data versions manually?

Firstly, I created 2 train files and a new raw file for a given raw file, where one train file is cleaned and the other is (cleaned + scaled). The new raw file is just a renaming of the old raw file for easier tracking of versions. The most difficult part of handling data versions manually was determining how to version the data itself. Here are some of the challenges I faced during manual data versioning:

1. **The vX idea:** My initial idea was to name a preprocessed file as **vX_train.csv**, where X is a number that I increment after each step of preprocessing. The problem with this approach is that if I gave you *v4_train.csv*, I don't know whether it is just cleaned or (cleaned + scaled) or even which raw file it belongs to. Although I can see the details in **manifest.txt**, I needed a more robust way to version so that understanding it would be easier.

2. **The vX.Y idea:** So I tried to name my file as **vX.Y_train.csv**, where X represents if data is cleaned or not, and Y represents if data is scaled or not. This was better than the previous approach, but the problem is that if I gave you *v1_1_train.csv*, I know it is (cleaned + scaled), but I don't know which raw file it belongs to.

3. **The vX.Y.Z idea:** So I tried to name my file as **vX.Y.Z_cleaned.csv** and **vX.Y.Z_scaled.csv**, where X represents the raw data version, Y represents if data is cleaned or not, and Z represents if data is scaled or not. This is good, but how do I detect if the raw file has changed?

4. **Timestamps:** I thought different raw data would have different timestamps, so I tried to version it accordingly. However, the problem is that if I delete the last 10 rows and add the same 10 rows back, the data is the same, but the timestamp is different, so it would incorrectly create a new version.

5. **File Path:** The idea was basically that different raw data files have different file paths, but that's not always true. I can just add some more columns to the existing data file, which would make it new data, of course, but the path would remain the same.

5. Num Rows: So now I tried to check the number of rows in the data. If the number of rows changes, then the raw data has changed. I would increase the version of X by plus one. This seems fair, but the problem is that if I say I have deleted the last 10 rows of the data and added 10 new rows, the data has changed, but the length of rows remains the same. Although it is new data, it didn't create a new version.

6. Hashing: So, to create a perfect system, what I done was hash the CSV file using the MD5 technique. After hashing the file, I store the hashes in a file named **version_history.json**. If a hash is not present in that file, it is new data, so I will increase the X; otherwise, I won't. These are the challenges I have faced during manual data versioning.

2 Ensuring Model & Data Integrity

Q2: How did you ensure that the model in production was the same one you evaluated during training?

To ensure the model I am using in production was the one evaluated during training, I followed a particular naming schema for the model which is version_name_model.joblib. I used joblib to save the model and while training, I know which model version will be generated, and which algorithm I am using from config.yaml. So in deployment I updated its path in config.yaml in the format of version_algorithm_model.joblib.

Q3: How did you ensure that train.py always used the correct version of the data specified in config.yaml?

To ensure that **train.py** always uses the correct data version specified in **config.yaml**, I have created a variable called **train_data_file** in the config itself, and in train.py, I fetch the train path from the config . So, whenever the train file has been updated, I simply change its update its path in config.yaml, and train.py takes care the rest.

3 The Case for MLOps Automation

Q4: Based on this experience, what are the top three features you would want in an automated MLOps tool?

Some of the features i need in my automated MlOps tool are

1. **Model Loading:** In my Flask API loads the model into memory only once so if i found a better model i need to restart the server again manually i need my automated to automatically restart if there is a better model

2. **Model :** In phase B, I have to juggle with so many JSON files to check which model is better and manually update in config.yml file. So, in automated ML Ops, I need a central model registry which should contains all the information about all the information about models and their accuracies and their paths so that I don't have to juggle with all the files.

3. **Monitoring :** Right now in Phase D I'm manually creating drift and manually retraining the model on the updated data. So the tool needs to continuously monitor my model performance on the production data and if the production error rate exceeds a certain threshold or accuracy goes down by a certain threshold.It should automatically trigger retraining the model on the updated data.

4 Deployment & Monitoring

Q5: Describe the "breakdown" you experienced when trying to track which model version was currently serving requests in your API.

One of the major breakdown that I have experienced during the model version serving the api was that the Flask API loads the model into memory only once, at the start of the server. predictions are based on that model.say now I realized i got better model with better accuracy and precision so it wil be updated in config.yaml but the server still runs on previous model as it only loads the model on start on server so i need to manually restart the server.

Q6: How would an automated Model Registry have made Phase B easier?

Right now in Phase B I am manually checking which model is in train and which model is in production, what model was running and what model to pick based on accuracy logs. If I have a model registry, I can automate this process very simply as it's a centralized repository that contains all of this information. So, the if there is a model registry then i can check which models are production and whuch models are having higheest accuracy and make decision on at that its lika a one step solution for all.