# MLOPS Assignment 1

## Group Members:

1.

2.

3.

4.

## **M2: Process and Tooling**

**Objective**: Gain hands-on experience with popular MLOps tools and understand the processes they support.

**Tasks**:

1. **Experiment Tracking**:

• Use MLflow to track experiments for a machine learning project.

• Record metrics, parameters, and results of at least three different model training runs.

2. **Data Versioning**:

• Use DVC (Data Version Control) to version control a dataset used in your project.

• Show how to revert to a previous version of the dataset.

**Deliverables**:

• MLflow experiment logs with different runs and their results.

• A DVC repository showing different versions of the dataset.

1. **Experiment Tracking using MLflow:**

**Step 1: Install required packages**

pip install mlflow

**Step 2: Set Up MLflow Tracking**

**# Import necessary packages**

import mlflow

import mlflow.sklearn

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error

import os

from pathlib import Path

**# Set the tracking URI to the current directory**

tracking\_uri = Path(os.getcwd()).as\_uri()

mlflow.set\_tracking\_uri(f"{tracking\_uri}/mlruns")

**# Load the dataset**

data = pd.read\_csv('age\_prediction\_dataset.csv')

**# One-hot encode categorical features**

data = pd.get\_dummies(data, columns=['Age\_group', 'Gender', 'PAQ605', 'Diabetic or not', "Respondent's Oral"])

**# Split the dataset for training and testing**

X = data.drop(['ID', 'Age'], axis=1)

y = data['Age']  # Target column

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

**Step 3: Track Experiments**

Train the Random Forest Regressor model and log metrics, parameters, and results.

**# Function to train and log the model**

def train\_and\_log\_model(n\_estimators, max\_depth, learning\_rate, epochs):

    with mlflow.start\_run():

**# Log parameters**

        mlflow.log\_param("n\_estimators", n\_estimators)

        mlflow.log\_param("max\_depth", max\_depth)

        mlflow.log\_param("learning\_rate", learning\_rate)

        mlflow.log\_param("epochs", epochs)

**# Train the model**

        model = RandomForestRegressor(n\_estimators=n\_estimators, max\_depth=max\_depth, random\_state=42)

        model.fit(X\_train, y\_train)

**# Predict and evaluate the model**

        predictions = model.predict(X\_test)

        mse = mean\_squared\_error(y\_test, predictions)

**# Log the model and metrics**

        mlflow.sklearn.log\_model(model, "model")

        mlflow.log\_metric("mse", mse)

        print(f"Run with n\_estimators={n\_estimators}, max\_depth={max\_depth}, mse={mse}")

**# Run experiments with different values of n\_estimators, max\_depth, learning\_rate, and epochs**

train\_and\_log\_model(n\_estimators=100, max\_depth=5, learning\_rate=0.01, epochs=200)

train\_and\_log\_model(n\_estimators=200, max\_depth=10, learning\_rate=0.1, epochs=20)

train\_and\_log\_model(n\_estimators=300, max\_depth=15, learning\_rate=0.5, epochs=100)

Once this python script is ran using the command

“python mlflow\_tracking.py”, it creates **mlruns** directory and logs metrices, parameters and accuracy for all the runs.

**Directory structure**:

It creates a separate directory for each run.

A screenshot of a computer

Description automatically generated

And for each run it creates artifacts, metrics, params, tags directory and logs accordingly.

A screenshot of a computer program

Description automatically generated

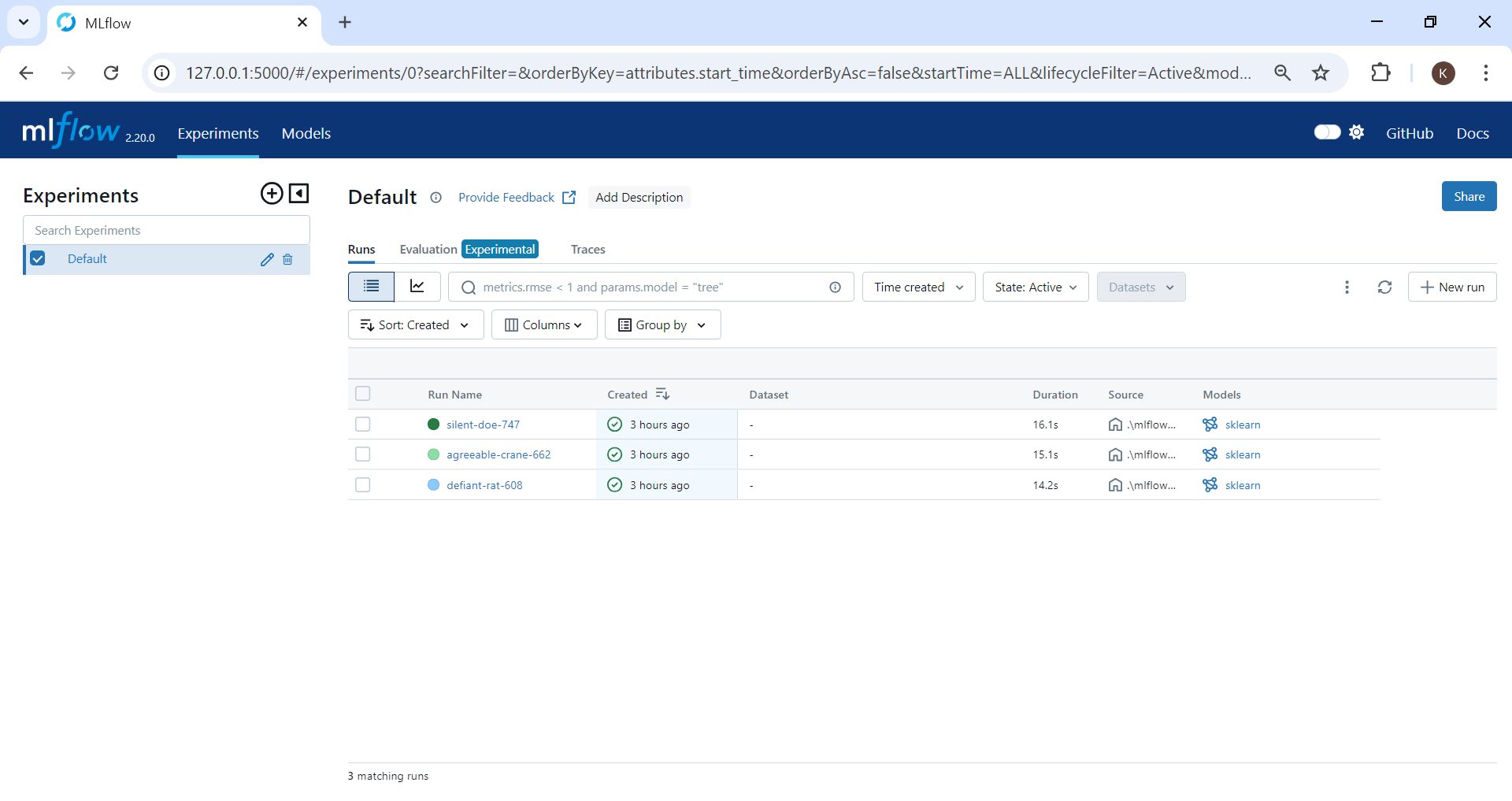
**To track the runs using mlflow ui:**

**Step 1: Configure MLflow Tracking Server:**

- Start the server: mlflow ui

- Access the UI at <http://localhost:5000>

We can also view the logs including metrics, parameters, and results using mlflow ui. Once the server is started, we can view the runs in the default section as below.



To view the result of each run, we can click on the run name.

**Run – 1:**

**Overview:**

In the overview section, we can see the Details, Parameters, Metrics and accuracy for each run.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

**Model Metrics:**

A blue and white rectangle with a white background

Description automatically generated

**Run – 2:**

**Overview:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Model Metrics:**

A blue and white rectangle with a white stripe

Description automatically generated

**Run – 3:**

**Overview:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Model Metrics:**

A blue and white bar graph

Description automatically generated

We can also compare 3 runs by selecting all runs in the **“Evaluation”** tab and clicking **“Compare”** button.

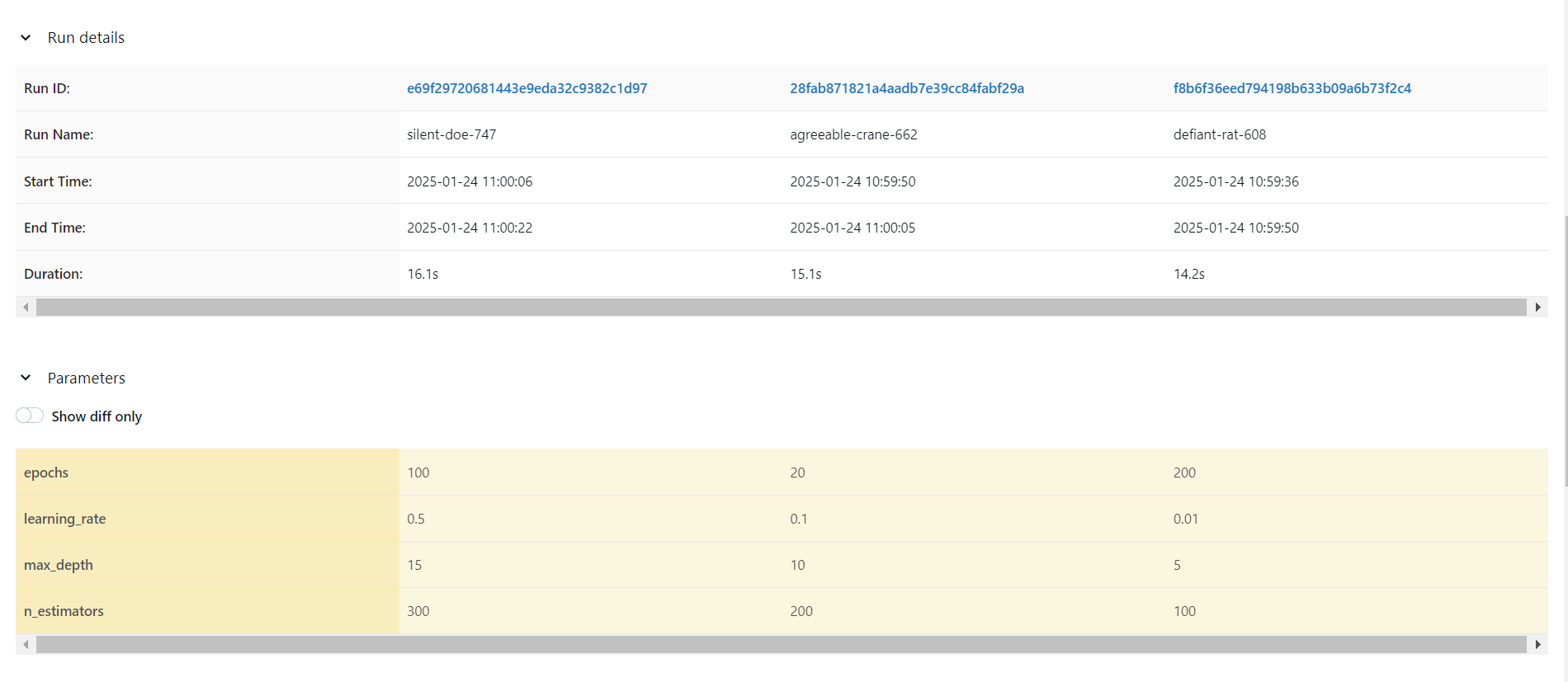
A screenshot of a computer

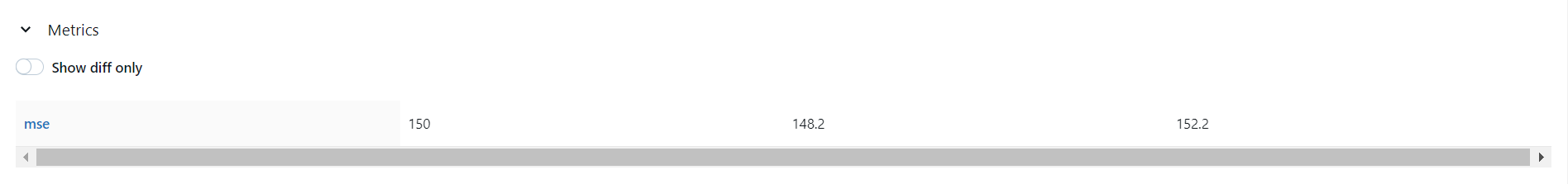
Description automatically generated

Once compare button is pressed, we can visualize the comparison in different ways.

A screenshot of a computer screen

Description automatically generated





1. **Data Versioning using DVC:**

**Step 1: Install required packages**

pip install dvc

**Step 2: Initialize DVC**

git init

dvc init

**Step 3: Add Dataset to DVC**

dvc add age\_prediction\_dataset.csv

**Step 4: Commit Changes**

git add .

git commit -m "Add dataset version 1"

**Step 5: Version Control the Dataset**

After modifying the dataset by adding more data to the dataset, we commit those changes.

dvc add age\_prediction\_dataset.csv

git add .

git commit -m "Update dataset to version 2"

**Step 6: Revert to a Previous Version**

dvc checkout age\_prediction\_dataset.csv

**Step 7: Check Status**

dvc status