Lab2 - MLOps

This lab has 3 sections:

- MLFLOW
- Logging
- Auto model calibration and versioning.

Before starting the lab be sure to follow these steps:

Step 1: Creating a Virtual Environment

In software development, it's crucial to manage project dependencies and isolate your project's environment from the global Python environment. This isolation ensures that your project remains consistent, stable, and free from conflicts with other Python packages or projects. To achieve this, we create a virtual environment dedicated to our project.

To create a virtual environment, follow these steps:

1. Open a Command Prompt or Terminal in the directory where you want to create your project.
2. Choose a name for your virtual environment (e.g "lab_02") and run the appropriate command:
    - `python -m venv lab02`

3. Activate the virtual environment (be sure to run this within the command line)

    - `Linux/Mac: source ./lab02/Scripts/activate`
    - `Windows: lab02\Scripts\activate`

After activation, you will see the virtual environment's name in your command prompt or terminal, indicating that you are working within the virtual environment.

Step 2: Creating a GitHub Repository, Cloning and Folder Structure

Now that we have set up our virtual environment, the next step is to create a GitHub repository for our project and establish a structured folder layout. This organization helps maintain your project's code, data, and tests in an organized manner.

Creating a GitHub Repository

- Open a web browser and go to GitHub.
- In the upper right corner, click the "+" button and select "New repository."
- Choose a name for your repository.
- Choose the visibility of your repository—either public (visible to everyone) or private (accessible only to selected collaborators)
- Check the "Initialize this repository with a README" box. This will create an initial README file that you can edit to provide project documentation.
- Click the "Create repository" button.

Cloning the Repository

- Open a Command Prompt or Terminal.
- Navigate to the directory where you want to clone your GitHub repository. This should be the same directory where you created your virtual environment.
- Run the following command to clone your GitHub repository into the current directory:
    - `git clone <repository_url>`

Replace <repository_url> with the URL of your GitHub repository. You can find this URL on your GitHub repository's main page. After running the command, the repository will be cloned, and you'll have a local copy of your GitHub project in your chosen directory.

Establishing Folder Structure

- Once you have cloned your repository, you can establish a structured folder layout within it. This layout helps organize your project into key directories for code, data, and tests. Create the following subfolders within your repository:
- data: This folder is used for storing project data files or datasets.
- src: This folder is where you'll store your project's source code files.
- test: This folder is dedicated to unit tests and test scripts for your code.
- Create a file named .gitignore. This is useful to exclude the virtual environment and other unnecessary files from version control.

- Create a requirements.txt file within the main folder of your local repository. Add the following packages to this file: scikit-learn, numpy, pandas, pytest, ipykernel, mlflow
- Add the virtual environment folder name inside your gitignore file so that it's not tracked by Git.

Adding and Pushing Your Project Code to GitHub

Now that we have our virtual environment set up, the GitHub repository created, and the folder structure organized, let's add our project's code and push it to GitHub.

Adding Your Project Code

- Navigate to your project directory using the Command Prompt or Terminal, where you have the virtual environment and folder structure set up.
- Create and write your Python code or other project files within the specified directories (src, data, etc.) according to your project requirements.
- Once your project files are ready, it's time to add them to Git's staging area. In your project directory, run the following command:
  - `git add .`
- This command stages all the changes and new files in your project directory for the next commit.

Committing Your Changes

- After staging your changes, commit them with a meaningful commit message that describes the changes you made. Replace <your_commit_message> with a descriptive message:

  - `git commit -m "<your_commit_message>"`

Pushing to GitHub

- To push your committed changes to your GitHub repository, use the following command:
  - `git push origin main`

Install the necessary packages within the requirements.txt by running:

- Pip3 install -r requirements.txt