the flow should be like : proejct scoping , data pipeline assignment , model developemnt pipleline and then model development

Here is the same combined guidelines document, reordered as you requested: Project Scoping → Data Pipeline → Model Development → Deployment.[1] [2] [3] [4]

Project Scoping Submission – [Project Name]

Team Members:

- [Team Member 1]
- [Team Member 2]
- [Team Member 3]
- [Team Member 4]
- [Team Member 5]

1. Introduction
   ➔ In this section, introduce the project and provide a brief overview.
2. Dataset Information
3. Dataset Introduction: Describe the dataset you will be working with, including its purpose and relevance to the project.
4. Data Card: Create a data card summarizing key details about the dataset, such as size, format, and data types.
5. Data Sources: Provide information about the sources of the data, including URLs, APIs, or any relevant references.
6. Data Rights and Privacy: Explain the rights and privacy considerations related to the dataset, including any compliance with data protection regulations (e.g., GDPR).
7. Data Planning and Splits
   ➔ Outline your data preprocessing steps and splitting strategies. (Loading, Preprocessing, Managing data)
8. GitHub Repository
   ➔ Share the link to your GitHub repository and describe the folder structure.

● README: Include a README file with essential project information, installation instructions, and usage guidelines.

5. Project Scope
   ➔ Follow the structured approach learned in class to break down the project:
6. Problems: List the main problems your project aims to address.
7. Current Solutions: Describe existing solutions or approaches to these problems.
8. Proposed Solutions: Present your proposed solutions and innovations.
9. Current Approach Flow Chart and Bottleneck Detection
   ➔ Visualize the current approach using a flowchart and identify potential bottlenecks. Discuss how you could improve the current process.
10. Metrics, Objectives, and Business Goals
    ➔ Define the key metrics you'll use to evaluate your project's success. Outline the project's objectives and how they align with broader business goals.
11. Failure Analysis
    ➔ Discuss potential risks, including what could go wrong during the project and after deployment, and provide an analysis of pipeline failures and mitigation strategies.
12. Deployment Infrastructure
    ➔ Provide detailed information about the infrastructure required to deploy your project, along with a list of supported platforms. Be sure to include necessary flowchart diagrams.
13. Monitoring Plan
    ➔ Provide a broad description of your monitoring plan, including what you intend to monitor and why. Prepare for detailed documentation.
14. Success and Acceptance Criteria
    ➔ Define the criteria for success and acceptance of the project.
15. Timeline Planning
    ➔ Create a preliminary project timeline, which can be modified based on given deadlines and constraints.
16. Additional Information
    ➔ Include any other relevant information you believe is necessary for a comprehensive project scoping submission.

MLOps Course: Data Pipeline Submission Guidelines

1 Overview
For the next phase of your project, you are required to submit a well-structured data pipeline in DAG format. The pipeline should be built using Airflow (or a similar orchestration tool) and must cover all essential stages from data acquisition to preprocessing, testing, versioning, and workflow management. Follow the detailed guidelines below:

2 Key Components to Include in Your Data Pipeline

1. Data Acquisition:
   Write code to download or fetch data from the necessary sources (APIs, databases, etc.). Ensure this step is reproducible, specifying any external dependencies in the requirements.txt or environment.yml file.

2. Data Preprocessing:
   Include clear steps for data cleaning, transformation, and feature engineering. Your preprocessing code should be modular and reusable, enabling easy adjustment.
3. Test Modules:
   Write unit tests for each component of your pipeline, especially for data preprocessing and transformation. Use testing frameworks such as pytest or unittest to ensure robustness. Test for edge cases, missing values, or possible anomalies.
4. Pipeline Orchestration (Airflow DAGs):
   Structure your pipeline using Airflow DAGs, ensuring logical connections between tasks. Airflow should handle the entire workflow, from downloading data to generating final outputs.
5. Data Versioning with DVC:
   Use Data Version Control (DVC) to track and version control your data. Include the relevant .dvc files for datasets, and ensure that Git tracks your code and configurations for reproducibility.
6. Tracking and Logging:
   Incorporate logging into your pipeline to track progress. Use Python's logging library or Airflow's built-in logging. Set up monitoring for anomalies and alerts when errors are detected.
7. Data Schema & Statistics Generation:
   Automate the generation of data schema and statistics using tools such as MLMD or TensorFlow Data Validation (TFDV) or DAG. Ensure that schema and data quality are validated over time.
8. Anomaly Detection & Alerts:
   Set up mechanisms to detect data anomalies such as missing values, outliers, or invalid formats. Ensure your pipeline triggers an alert (e.g., email, Slack) in case of anomalies.
9. Pipeline Flow Optimization:
   Use Airflow's Gantt chart to identify bottlenecks in your pipeline. Optimize slow tasks by parallelizing or improving performance.

3 Data Bias Detection Using Data Slicing

1. Detecting Bias in Your Data:
   To ensure that your data is not biased, you need to perform data slicing and analyze performance across different subgroups. Identify demographic or categorical features in your data (such as age, gender, location, etc.) and evaluate how your model behaves for each of these subgroups.
2. Data Slicing for Bias Analysis:
   Use tools such as SliceFinder, TensorFlow Model Analysis (TFMA), or Fairlearn to implement data slicing techniques. These tools can help you split your data into meaningful slices and track model performance on each slice. This helps you determine whether the model is biased toward or against certain groups.
3. Mitigation of Bias:
   If you detect any significant performance differences across slices, you must take steps to

mitigate bias. Techniques such as re-sampling underrepresented groups, applying fairness constraints, or adjusting decision thresholds can be used to address this.

4. Document Bias Mitigation Process:
   Clearly document the steps you took to detect and mitigate bias. Include explanations of the types of bias found, how you addressed them, and whether any trade-offs were made in terms of overall performance.

4 Additional Guidelines

1. Folder Structure:
   Ensure your project is well-organized. Below is an example structure:

```
/Project Repo
|- Data-Pipeline/
|- dags/
|- data/
|- scripts/
|- tests/
|- logs/
|- dvc.yaml
-- README.md
```

2. README Documentation:
   Your GitHub repository must include a detailed README.md file, which should contain:

- Instructions for setting up the environment.
- Steps to run the pipeline.
- Code structure explanation.
- Reproducibility details and data versioning with DVC.

3. Reproducibility:
   Provide clear steps in the README.md file to replicate your pipeline on other machines. Anyone should be able to clone the repository, install dependencies, and run the pipeline without errors.
4. Code Style:
   Follow modular programming practices and ensure your code adheres to Python's PEP 8 guidelines. Ensure clarity, modularity, and maintainability in your code.
5. Error Handling & Logging:
   Implement error handling for potential failure points in the pipeline (e.g., data unavailability, file corruption). Ensure logs provide sufficient information for troubleshooting.

5 Evaluation Criteria

You will be evaluated based on the following criteria:

1. Proper Documentation:
   Your GitHub repository should have a clear README, well-commented code, and a structured folder organization.

2. Modular Syntax and Code:
   Each component of your pipeline should be written in a modular and reusable format, allowing for easy updates and testing.
3. Pipeline Orchestration (Airflow DAGs):
   Your data pipeline must be implemented using Airflow DAGs (or a similar orchestration tool). The flow of tasks should be logical and error handling should be properly implemented.
4. Tracking and Logging:
   Ensure proper tracking of tasks and incorporate logging throughout the pipeline. Set up error alerts and notifications for anomalies or failures.
5. Data Version Control (DVC):
   Properly manage your data and models using DVC. Ensure that data is versioned and that the history of changes is maintained alongside your code in Git.
6. Pipeline Flow Optimization:
   Use tools such as Airflow Gantt charts to identify and resolve bottlenecks in the pipeline, optimizing task performance and overall flow.
7. Schema and Statistics Generation:
   Automatically generate data schema and statistics using tools like Great Expectations or TFDV. You will be evaluated on how well you validate the quality of your data.
8. Anomalies Detection and Alert Generation:
   Implement mechanisms to detect data anomalies and generate alerts. You should be able to handle issues such as missing values, outliers, and schema violations.
9. Bias Detection and Mitigation:
   You will be assessed on your ability to detect and mitigate bias in the dataset through data slicing. Ensure that your model performs equitably across different subgroups and that bias is addressed in your report.
10. Test Modules:
    Unit tests should be included for each key component of your pipeline, particularly for data preprocessing and transformation steps. This ensures robustness and allows testing for edge cases.
11. Reproducibility:
    Your entire pipeline should be easily reproducible on another machine. This includes instructions on setting up the environment and running the pipeline, without any errors.
12. Error Handling and Logging:
    Ensure that your pipeline has robust error handling for potential failure points. Logs should provide sufficient information to troubleshoot issues. Every step of the pipeline must be carefully designed, documented, and tested to ensure functionality, robustness, and reproducibility.

Model Development Guidelines

1 Overview
This section of your project focuses on the development of the machine learning (ML) model. Depending on your project, you might be developing a new model or fine-tuning a pre-trained

large model. The following guidelines cover model development, hyperparameter tuning, experiment tracking, validation, sensitivity analysis, bias detection, and CI/CD pipeline automation.

2 Model Development and ML Code

Your model development process should include the following key elements:

1. Loading Data from the Data Pipeline:
   Use code to load data that has been processed and versioned through the data pipeline. Ensure that this step integrates seamlessly with the output of the pipeline.
2. Training and Selecting the Best Model:
   Develop the code to train your model. Depending on your task, you may need to try different model architectures or configurations. Ensure your code includes logic for selecting the best model based on performance metrics.
3. Model Validation:
   Implement a model validation process that includes performance metrics relevant to your task (e.g., accuracy, precision, recall, F1-score, AUC). Validation should be performed on a hold-out dataset that was not used during training.
4. Model Bias Detection (Using Slicing Techniques):
   To ensure fairness, evaluate your model across different slices of the data, such as demographic groups, to detect bias. Use tools such as Fairlearn or TensorFlow Model Analysis (TFMA) to perform data slicing and check if the model behaves equitably across different subgroups.
5. Code to Check for Bias:
   Your code should include functionality for running bias checks, specifically across different slices of the dataset, and generate reports on any performance disparities. If bias is detected, suggest mitigation strategies, such as re-sampling or fairness constraints.
6. Pushing the Model to Artifact or Model Registry:
   Once the best model is selected and validated (with bias checks completed), push the model to a model registry such as Google Cloud Artifact Registry (GCP) for version control and reproducibility.

3 Hyperparameter Tuning

If applicable, perform hyperparameter tuning to optimize the model’s performance. Use techniques such as grid search, random search, or Bayesian optimization to find the best set of parameters. Be sure to document the search space and the tuning process in your submission.

4 Experiment Tracking and Results

You are expected to track your model development experiments and decisions throughout the project. This can include:

- Tracking Tools:
  Use tools like MLflow or Weights & Biases to track your experiments and results. Each run should log hyperparameters, model performance metrics, and model versions.

- Results and Model Selection:
  Provide visualizations or images showing the breakdown of results (e.g., bar plots, confusion matrices). These images should highlight the comparison between different models and explain how the final model was selected.

5 Model Sensitivity Analysis

Perform sensitivity analysis to determine how the model's performance changes with respect to different input features or hyperparameters. You can use techniques like:

- Feature Importance Analysis:
  Use methods such as SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-Agnostic Explanations) to evaluate how different features impact the model's predictions.
- Hyperparameter Sensitivity:
  Analyze how changes in hyperparameters affect model performance. This can help identify which hyperparameters have the most significant effect on the final outcomes.

6 Model Bias Detection (Using Slicing Techniques)

It is important to ensure that your model does not exhibit bias toward or against any subgroup of the population. You should:

1. Perform Slicing:
   Break down the dataset by meaningful slices, such as demographic groups, and evaluate the model performance on each slice.
2. Track Metrics Across Slices:
   Track key metrics (e.g., accuracy, F1-score) across these slices and report on any significant disparities.
3. Bias Mitigation:
   If bias is detected, apply techniques to mitigate it. This may include re-weighting, re-sampling, or adjusting decision thresholds for different groups.
4. Document Bias Mitigation:
   Clearly document the steps taken to detect and address bias, and explain any trade-offs made in the process.

7 CI/CD Pipeline Automation for Model Development

To ensure automation in your model development process, implement a continuous integration and continuous deployment (CI/CD) pipeline. This pipeline should trigger model training and validation whenever new code is pushed that affects the model. Key steps include:

1. CI/CD Setup for Model Training:
   Configure a CI/CD pipeline (e.g., using GitHub Actions, Jenkins, or Google Cloud Build) to automate model training. When new code is committed to the repository, the pipeline should trigger model training, either locally or on Google Cloud Platform (GCP), based on your setup.

2. Automated Model Validation:
   After the model training, the pipeline should automatically validate the trained model by evaluating its performance on the validation set and generating metrics. If the validation passes a predefined threshold, the model proceeds to the next step.
3. Automated Model Bias Detection:
   The pipeline should also include steps to perform bias detection across different data slices. Results from these checks should be logged, and any significant bias should trigger an alert or block the deployment.
4. Model Deployment or Registry Push:
   Once the model passes validation and bias detection, it should be automatically pushed to the model registry (e.g., GCP Artifact Registry or Model Registry). If your project includes a deployment, the pipeline should also handle the deployment of the model to production environments.
5. Notifications and Alerts:
   Set up notifications (e.g., via email or Slack) for pipeline failures, training completion, or if the model fails validation or bias checks. This ensures that you are aware of the current state of your model pipeline.
6. Rollback Mechanism:
   Implement a rollback mechanism in case the newly trained model performs worse than the previous model. This ensures that a suboptimal model is not deployed into production.

8 Code Implementation

Your code implementation for model development should include:

1. Docker or RAG Format:
   Implement the entire model development process in either Docker format (containerizing your code) or a retrieval-augmented generation (RAG) system. This ensures reproducibility and portability of your ML models.
2. Code for Loading Data from Data Pipeline:
   Write code that loads the data output from the data pipeline, ensuring all transformations and versioning are applied as needed.
3. Code for Training Model and Selecting Best Model:
   Include code that trains the model and selects the best model based on validation metrics. This can be done by comparing performance across multiple model architectures.
4. Code for Model Validation:
   Write code that validates the model on a separate validation dataset and computes metrics relevant to your task.
5. Code for Bias Checking:
   Develop code to check for bias using data slicing techniques. Generate bias reports and visualizations to demonstrate your model's fairness across different groups.
6. Code for Model Selection after Bias Checking:
   Ensure that the final model selection is done after considering both validation performance and bias analysis results.

7. Code to Push Model to Artifact Registry/Model Registry:
   Once the best model is selected and validated, include code to push it to an artifact or model registry on GCP for versioning and future use.

Note: If you are using pre-trained large models, some of these steps might not be necessary (e.g., model training or hyperparameter tuning). However, you must still perform validation, bias detection, and tracking.

Model Deployment Submission Guidelines

1 Overview
In this section of your project, you are required to deploy the machine learning model that you have developed. Depending on whether your deployment is cloud-based or edge-based, you will need to select an appropriate deployment strategy. Your submission must include all necessary steps and code to automate the deployment process, along with detailed instructions for replication. Additionally, you are required to submit a video demonstrating the entire deployment process on a fresh environment, without any prior installation.

2 Cloud vs. Edge Deployment

Your first step is to specify whether your model will be deployed on the cloud or at the edge:

- Cloud Deployment:
  If your model is deployed in the cloud, you must specify the cloud provider (e.g., Google Cloud Platform, AWS, Azure) and the service being used for deployment (e.g., Kubernetes, Cloud Functions, Vertex AI).
- Edge Deployment:
  If deploying at the edge (e.g., IoT devices, mobile devices, or edge servers), you must explain your edge deployment strategy and provide details on how the model will be optimized for lower resource environments.

3 Cloud Deployment (GCP Example)

If you are using Google Cloud Platform (GCP) for cloud deployment, you need to specify the following details:

1. Deployment Service:
   Indicate the service used for deployment, such as:

- Kubernetes (GKE):
  Use Google Kubernetes Engine for containerized deployment. Provide details on the configuration of pods, services, and load balancing.
- Cloud Functions:
  Use Google Cloud Functions for serverless deployment. Include triggers (e.g., HTTP requests) and how to connect the model for inference.
- Vertex AI:
  Use Vertex AI for model serving, including steps for deployment of trained models, model versioning, and the pipeline for continuous integration and deployment.

2. Deployment Automation:
   Your submission must include automated deployment scripts (e.g., using Terraform, Cloud Build, or CI/CD pipelines) to ensure that deployment is not a manual process. These scripts should automatically:

- Pull the latest model from the repository or model registry.
- Deploy or update the model to the selected service.
- Monitor and log the deployment status.

3. Connection to Repository:
   Include code that connects your deployment service to your repository (e.g., GitHub) to automatically trigger redeployments when new model versions are pushed. Use tools like GitHub Actions, Google Cloud Build, or Jenkins to create a CI/CD pipeline.
4. Detailed Steps for Replication:
   Provide clear, step-by-step instructions to replicate your deployment process. These steps should include:

- Instructions for setting up the environment (e.g., dependencies, permissions).
- Instructions for running your deployment scripts.
- Instructions for verifying the deployment (e.g., accessing a deployed endpoint for model inference).

4 Edge Deployment

For edge deployment, your model must be optimized for deployment in resource-constrained environments. The following aspects must be included:

1. Model Optimization:
   Provide details on how the model has been optimized for the edge, including techniques like model quantization, pruning, or knowledge distillation. This ensures the model can run efficiently on low-power devices.
2. Deployment Platform:
   Specify the edge platform being used (e.g., TensorFlow Lite, ONNX, NVIDIA Jetson, or custom hardware) and provide steps for deployment on these platforms.
3. Connection to Repository and Deployment Automation:
   Include automation scripts that push the optimized model to the edge device. These scripts should also handle the process of pulling the model from the repository and deploying it to the edge device automatically.
4. Detailed Steps for Replication:
   Provide clear instructions to replicate the deployment process on other edge devices. These should include:

- Setup and configuration of the edge device.
- Installation of necessary libraries or frameworks (e.g., TensorFlow Lite).
- Steps for uploading and running the model on the edge device.

5 Model Monitoring and Triggering Retraining

Once your model is deployed, it is important to monitor its performance in real-time to ensure that it continues to perform well as new data comes in. Over time, models may suffer from model decay or encounter data drift (shifts in the underlying data distribution). Your model pipeline must include the following:

1. Monitoring for Model Decay and Data Shift:
   Implement a system to monitor key performance metrics (e.g., accuracy, F1-score, precision, recall) over time. Use these metrics to detect whether the model performance degrades or if there are signs of data shift (e.g., changes in input data distribution). You can use monitoring tools like Prometheus, Grafana, or Google Cloud Monitoring.
2. Detecting Data Shift:
   Include mechanisms to detect data shift, such as monitoring the input data distribution and comparing it with the training data distribution. Tools like Evidently AI or TensorFlow Data Validation (TFDV) can help with detecting data drift by tracking features and their distributions.
3. Threshold for Triggering Retraining:
   Set predefined thresholds for model performance or data shift. If the model's performance drops below the threshold or significant data drift is detected, the system should automatically trigger the retraining of the model. This can be done by integrating the monitoring system with your CI/CD pipeline to rerun the model pipeline and retrain the model.
4. Automating the Retraining Pipeline:
   When a data shift or model decay is detected, your pipeline should automatically pull the latest data, retrain the model, and redeploy the updated model. Ensure that your CI/CD pipeline includes this automation, with steps to:

- Pull new data from the source.
- Run the training pipeline to retrain the model.
- Validate and test the new model.
- If the new model performs better, deploy it to production.
- If not, maintain the existing model in production.

5. Notifications for Model Retraining:
   Set up notifications (e.g., via email or Slack) when the retraining process is triggered or when a new model is deployed. This helps ensure that stakeholders are aware of changes in the model's performance and retraining status.

6 Code Submission

Your code submission must include the following:

1. Deployment Scripts:
   Automated scripts for deploying the model, whether on the cloud or edge. These scripts should be structured in a way that anyone can run them with minimal manual intervention.
2. Connection to Repository:
   Code that links the repository with the deployment environment. For cloud deployments, this

includes setting up CI/CD pipelines to trigger automatic deployments when changes are made to the model code or repository.

3. Monitoring and Triggering Retraining:
   Include code for monitoring the deployed model and detecting decay or data shift. Additionally, provide scripts to trigger the retraining process and redeploy the updated model when necessary.
4. Environment Configuration:
   Provide all necessary environment configuration files (e.g., docker-compose.yaml, k8s.yaml, or edge deployment scripts) so that the environment can be easily replicated.
5. Logs and Monitoring:
   Ensure that your deployment includes logging and monitoring functionality to track the status of the model post-deployment. For cloud deployments, consider using Google Cloud Monitoring or similar tools for real-time monitoring.

7 Video Submission

In addition to the code and documentation, you are required to submit a video demonstrating the deployment process:

1. Recording on a Fresh Environment:
   The video should showcase how the model is deployed in a fresh environment, without any prior installations. Start from a clean machine and go through the entire process, showing how the environment is set up and the deployment is executed.
2. Step-by-Step Walkthrough:
   The video should clearly demonstrate the following steps:

- Environment setup and installation of dependencies.
- Running the deployment scripts (automated) to deploy the model.
- Verification of deployment, such as accessing the deployed model endpoint for inference.

3. Video Length and Quality:
   The video should be concise, ideally between 5–10 minutes, and should clearly demonstrate the entire process. Ensure high-quality audio and screen recording to make it easy to follow.

8 Evaluation Criteria

Your submission will be evaluated based on the following:

1. Correctness and Completeness:
   The deployment process must be fully automated and work as expected without manual intervention.
2. Documentation and Replication Steps:
   Clear, detailed instructions for replicating the deployment on other machines or platforms. Ensure that these steps cover setup, deployment, and validation.
3. Model Optimization (For Edge Deployments):
   The model should be properly optimized for edge deployment if applicable, ensuring that it can run efficiently on resource-constrained devices.

4. Automated CI/CD Integration:
   Your CI/CD pipeline should be set up to automatically trigger deployment when new model versions are pushed to the repository.
5. Logs and Monitoring:
   Include appropriate logging and monitoring to ensure the deployment can be tracked, and any issues post-deployment can be quickly identified and resolved.
6. Model Monitoring and Retraining:
   Include monitoring for model decay or data shift, and demonstrate how the system automatically triggers retraining and redeployment when performance deteriorates.
7. Video Demonstration:
   The video submission should clearly show the entire deployment process from a fresh environment, highlighting how everything works without any pre-existing installations.

⁂

1. data_pipeline___MLOPS-1-2-1.pdf
2. Project-Scoping.pdf
3. Deployment_pipeline.pdf
4. Model_pipeline.pdf