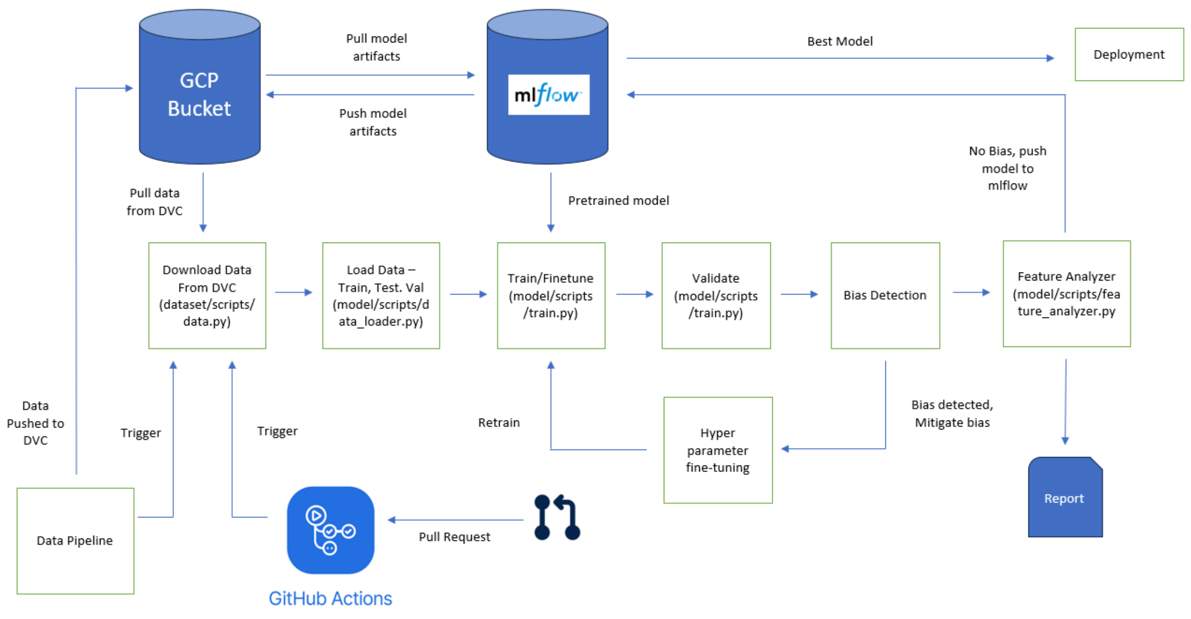
# Model Development Report

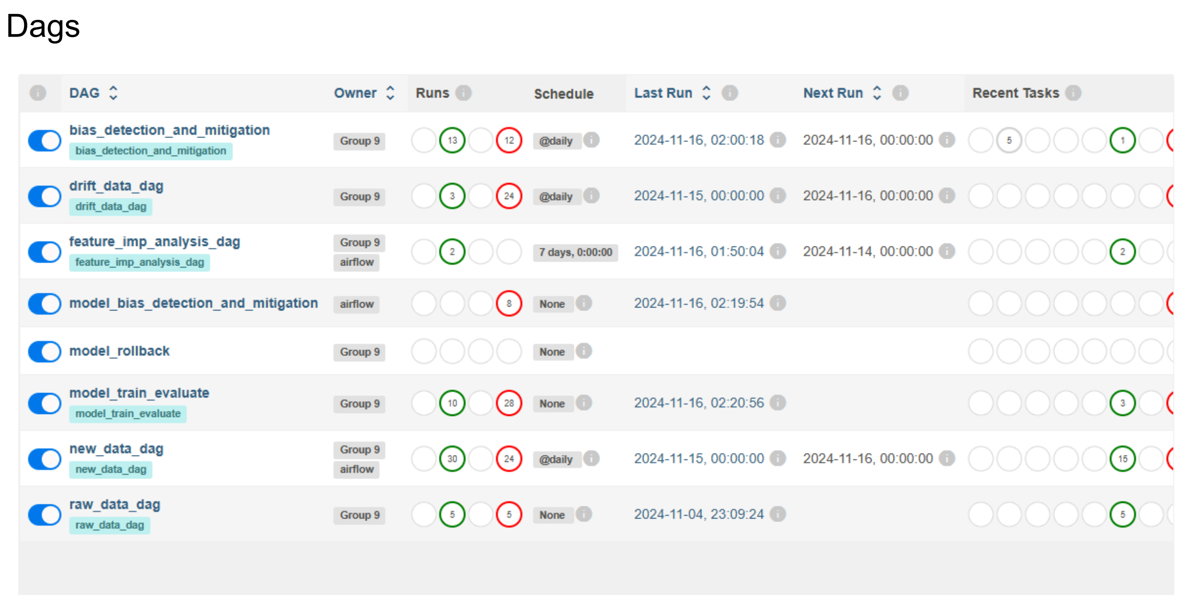
## 1. Overview


GCP Bucket
Pull model artifacts
Push model artifacts
mlflow
Best Model
Deployment
No Bias, push model to mlflow
Pull data from DVC
Pretrained model
Download Data From DVC (dataset/scripts/data.py)
Load Data – Train, Test. Val (model/scripts/data_loader.py)
Train/Finetune (model/scripts/train.py)
Validate (model/scripts/train.py)
Bias Detection
Feature Analyzer (model/scripts/feature_analyzer.py)
Data Pushed to DVC
Trigger
Trigger
Retrain
Hyper parameter fine-tuning
Bias detected, Mitigate bias
Report
Data Pipeline
Pull Request
GitHub Actions

MLFlow Server - [http://34.56.170.84:5000/](http://34.56.170.84:5000/)

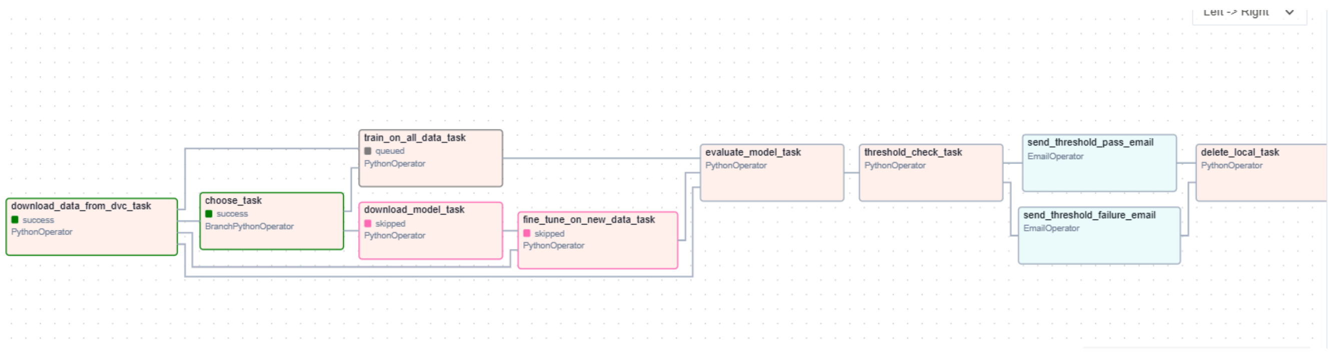
Code - [https://github.com/MLOPS-IE7374-Fall2024-G9/mlops-project](https://github.com/MLOPS-IE7374-Fall2024-G9/mlops-project)

The model pipeline employs MLflow for experiment tracking and hyperparameter tuning, allowing the team to log and monitor metrics across various training runs. The primary metrics used for model evaluation include RMSE, MAE, MSE, and R2, ensuring a comprehensive assessment of the model's performance on both accuracy and error minimization.

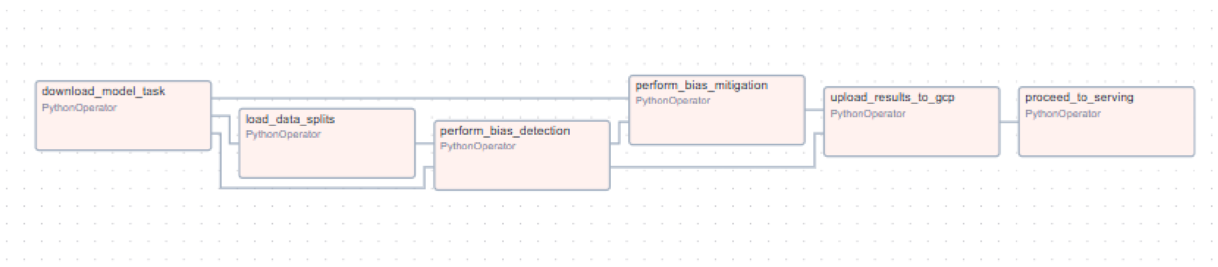


Dags

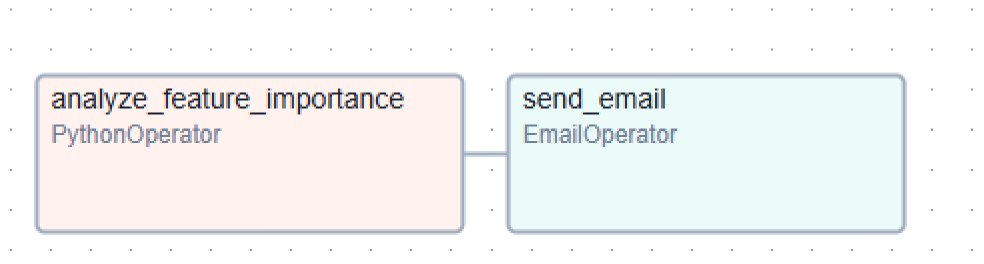
| DAG | Owner | Runs | Schedule | Last Run | Next Run | Recent Tasks |
|---|---|---|---|---|---|---|
| bias_detection_and_mitigation<br>bias_detection_and_mitigation | Group 9 | 13 12 | @daily | 2024-11-16, 02:00:18 | 2024-11-16, 00:00:00 | 5 1 |
| drift_data_dag<br>drift_data_dag | Group 9 | 3 24 | @daily | 2024-11-15, 00:00:00 | 2024-11-16, 00:00:00 | |
| feature_imp_analysis_dag<br>feature_imp_analysis_dag | Group 9<br>airflow | 2 | 7 days, 0:00:00 | 2024-11-16, 01:50:04 | 2024-11-14, 00:00:00 | 2 |
| model_bias_detection_and_mitigation | airflow | 8 | None | 2024-11-16, 02:19:54 | | |
| model_rollback | Group 9 | | None | | | |
| model_train_evaluate<br>model_train_evaluate | Group 9 | 10 28 | None | 2024-11-16, 02:20:56 | | 3 |
| new_data_dag<br>new_data_dag | Group 9<br>airflow | 30 24 | @daily | 2024-11-15, 00:00:00 | 2024-11-16, 00:00:00 | 15 |
| raw_data_dag<br>raw_data_dag | Group 9 | 5 5 | None | 2024-11-04, 23:09:24 | | 5 |



Model Pipeline DAG



Model Bias Detection DAG



Model Feature Importance DAG

## 2. Model Development and ML Code

1. **Loading Data from the Data Pipeline**: The data from the data pipeline is pushed to the DVC. When the data pipeline finishes execution, the model pipeline is triggered to train the model on the new dataset. The model is pulled from DVC and split into train, test, validation sets. Three models can be trained with these sets - XGBoost, Linear Regression and LSTM. In our case XGBoost is the selected best model. The data is downloaded and split using **model/data_loader.py,** and is saved in model/data folder.

2. **Training and Selecting the Best Model**: We experiment with multiple models and multiple algorithms. Specifically we are using random_search on XGBoost, with multiple configurable parameters and the best model is chosen. The parameters are present in **model/scripts/config.json.** The best model is selected automatically and pushed to mlflow server. The mlflow server saves the model artifacts in GCP

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.

   **Validation Workflow:**

   - **Cross-Validation**: The model undergoes 5-fold cross-validation during hyperparameter tuning to ensure robustness.
   - **Metrics Calculation**: Post-training, metrics such as RMSE, MAE, MSE, and R2 are calculated and logged in MLflow.
   - **Bias Detection**: After validation, a bias detection step is executed to examine fairness across different data slices.
   - **Threshold Validation**: Only models meeting the predefined performance threshold (based on validation metrics) proceed to the

next step in the pipeline.

4. **Model Bias Detection:**

   A model bias detection pipeline has been implemented to identify and mitigate bias injected into the code during the training phase. To identify the bias, we are splitting the data into groups based on the categorical features. In our dataset the only categorical feature that is present is ‘subba-name’ which represents multiple sub-regions in which we are trying to forecast load.

   To find whether the model has a bias in it, we perform a **Metric Frame Analysis** from the **FairLearn Library.** During this, we perform group analysis between multiple groups split by the sensitive feature aka categorical features.

   We use error metrics to identify how well the model predicts the target for different subgroups. This information helps us determine whether the model is biased.

   The metric we used is **Mean Absolute Error(MAE)** as the problem we are trying to solve is regression-based.

   We’ve set the threshold for each group to determine if there is a bias towards the particular group or not. We calculate the overall MAE for the whole data and see if any subgroups with absolute differences of their respective MAE and overall MAE are greater than **1.5 times the overall MAE**. If its greater, then there is a bias in it. With this approach, we are able to determine the subgroups which are affected by bias.

   After this step, we move forward to mitigate the bias in the model. For regression-based models, the most famous mitigation approaches are **resampling or re-weighting.**

   **Re-sampling** is the process of balancing the under-represented groups in the dataset by fetching new missing data or adding synthetic data. By default, the data sources are imperfect and do not provide an equal number of records per ‘subba-name’. So utilizing re-sampling would lead to adding mostly synthetic data for the model to be trained on.

   So, we decided to move forward with the model re-weighting technique, where the models are re-weighted to reduce the bias towards the sub-groups.

   Finally, after the mitigation step is performed, the re-weighted model is pushed further to be served.

5. **Pushing the Model to Artifact or Model Registry**:

   In our project, pushing the model to an artifact storage or model registry is a critical component of our MLOps workflow. We use a combination of **GCP Bucket** and **MLflow** to ensure efficient storage, tracking, and versioning of models throughout their lifecycle.

   1. **Artifact Storage with GCP Bucket**:
      After training and validating the model, we store the resulting model artifacts in a **GCP Bucket**. This serves as the centralized storage location for all our model files, ensuring accessibility and version control for future iterations or deployments.
   2. **Integration with MLflow**:
      We utilize **MLflow** as our model registry to track experiments, manage model versions, and record metadata, including:
      - Pretrained models used for fine-tuning.
      - Evaluation metrics like **RMSE**, **MAE**, **MSE**, and **R²**.
      - Tagging of the best-performing model for deployment.
   3. MLflow also allows us to easily retrieve the model for further testing or deployment to production environments.
   4. **Automated CI/CD Workflow**:
      - **GitHub Actions** is configured to automate the pipeline, ensuring any new changes to the codebase trigger model training and validation.
      - Once the validation and bias checks are complete, the model is automatically pushed to the **GCP Bucket** and registered in **MLflow** as part of our CI/CD pipeline..

## 4. Experiment Tracking and Results

- **Tracking Tools**: Used MLFlow with a self hosted server to track all our ML experiments and results.

## 5. Model Sensitivity Analysis

- **Feature Importance Analysis:**

  The FeatureImportanceAnalyzer class leverages SHAP (SHapley Additive exPlanations) to analyze the impact of individual features on a model's predictions, providing transparency into model behavior.

  This tool is essential for understanding which features contribute most significantly to predictions and detecting potential biases in feature importance.

The class loads a pre-trained model and a dataset, computes SHAP values using shap.TreeExplainer, and generates a summary plot that visually represents feature importance

- **Hyperparameter Sensitivity:**
  A hyperparameter sensitivity analysis is conducted to explore how different hyperparameter values affect model performance. This process is managed by RandomizedSearchCV in the training script, allowing for an efficient sweep of the parameter space. The most influential hyperparameters, such as `learning_rate`, `max_depth`, and `n_estimators`, are logged and analyzed through MLflow.

**Rollback Mechanism**:

The project focuses on **safe model deployment** and **version control** for machine learning systems. It ensures that if a newly deployed production model underperforms, the system can quickly revert to a previously validated version, minimizing the risk of adverse impacts on downstream applications.

Key Features and Processes:

1. **Model Versioning with MLflow**:
   The project uses **MLflow Model Registry** to track versions of a machine learning model. Each model version has associated metadata, such as its stage (`Production`, `Staging`, or `Archived`) and performance metrics.
2. **Production Model Management**:
   The system supports managing the lifecycle of models, focusing on identifying:
   - The current production model version.
   - The previous version for rollback purposes.
3. **Rollback Mechanism**:
   The main functionality ensures that if the latest production model fails or performs worse than expected, it can be replaced with the last known stable version:
   - **Current Production Model Identification**:
     The `get_current_production_model` function fetches the model currently tagged as "Production."
   - **Previous Model Retrieval**:
     The `get_previous_model` function determines the version immediately preceding the current production model.
   - **Stage Transition**:
     The `rollback_model` function:
     - Promotes the previous version to "Production."
     - Demotes the current version to "Archived" or another appropriate stage.
4. **Logging for Monitoring and Debugging**:

The system uses Python's `logging` module for tracking each step, making it easier to monitor and debug the rollback process.

## 6. CI/CD Pipeline Automation for Model Development

**Overview:**
A CI/CD pipeline is set up using GitHub Actions and Airflow to automate the training, validation, and deployment of models. This setup allows for a seamless integration between data versioning, model tracking, and deployment workflows.

**Pipeline Steps:**

1. **Data Pipeline Integration**:

   Data is pulled from DVC and processed into training, testing, and validation sets by Airflow DAGS. Any new data pushed to DVC triggers a data loading workflow.

2. **Automated Model Training**:

   The CI/CD pipeline triggers model training whenever a new pull request is merged. This training step leverages GitHub Actions to run Airflow DAGs that handle data loading, model training, and validation.

3. **Automated Model Validation**:

   After training, model performance on the validation set is automatically evaluated using metrics such as RMSE, MAE, MSE, and R2. If performance meets the specified threshold, the model proceeds to the next step; otherwise, retraining is triggered.

4. **Bias Detection and Mitigation**:

   The pipeline includes a bias detection phase where the model is evaluated for potential biases across different slices of data. If significant bias is detected, the pipeline stops and triggers an alert, prompting the need for mitigation steps.

5. **Artifact Storage**:

   Model artifacts, including trained models, are stored in a GCP Bucket and tracked via MLflow. This allows for version control and easy retrieval of past models.

6. **Model Registry and Deployment**:

   Validated models are pushed to the MLflow model registry. If deployment criteria are met, the model is automatically deployed to production, or else it is marked for retraining.

7. **Notifications and Alerts**:

   Airflow is configured to send email notifications for key events in the pipeline, such as training completion, validation failure, or bias detection alerts. This ensures continuous monitoring and prompt action when necessary