Reproducible Machine Learning Pipeline Using Dagster

Air Quality Index (AQI) Prediction

Objective

This assignment was aimed at creating a reproducible machine learning pipeline with Dagster that does not have to face the typical problems with rerunning Jupyter notebooks. The pipeline tracks the data dependencies, allows part re-execution, and provides reliable machine learning processes of the Air Quality Index (AQI) prediction among the Indian cities.

Dataset

It used Air Quality Data (India, 2015-2024) provided on Kaggle.[1] It has 18,265 records of air quality measurements of 5 big cities in India (Delhi, Mumbai, Chennai, Kolkata and Bangalore) over a period of 10 years. The samples consist of pollutant data (PM2.5, PM10, NO, NO2, NOx, NH3, CO, SO2, O3) and the dependent variable AQI (Air Quality Index).[1]

Pipeline Design

The pipeline was installed through the asset-based strategy of Dagster. Every logical step of the machine learning workflow was represented as an asset:

- **raw_data:** loads the AQI dataset in Google drive.
- **cleaned_data:** Clean data, removes missing values, and temporal extraction.
- **eda_results:** Produces 9 visualizations of comprehensive exploratory data analysis.
- **preprocessed_data:** Feature engineering and train-test split (80-20) is performed.
- **linear_regression model, decision tree model, random forest model, gradient boosting model:** Trains four models of machine learning separately.
- **model comparison:** Comparison and evaluation of all models based on R 2, RMSE and MAE.

Dagster automatically constructs a Directed Acyclic Graph (DAG) based on these dependencies, which is visualized in the Dagster UI.

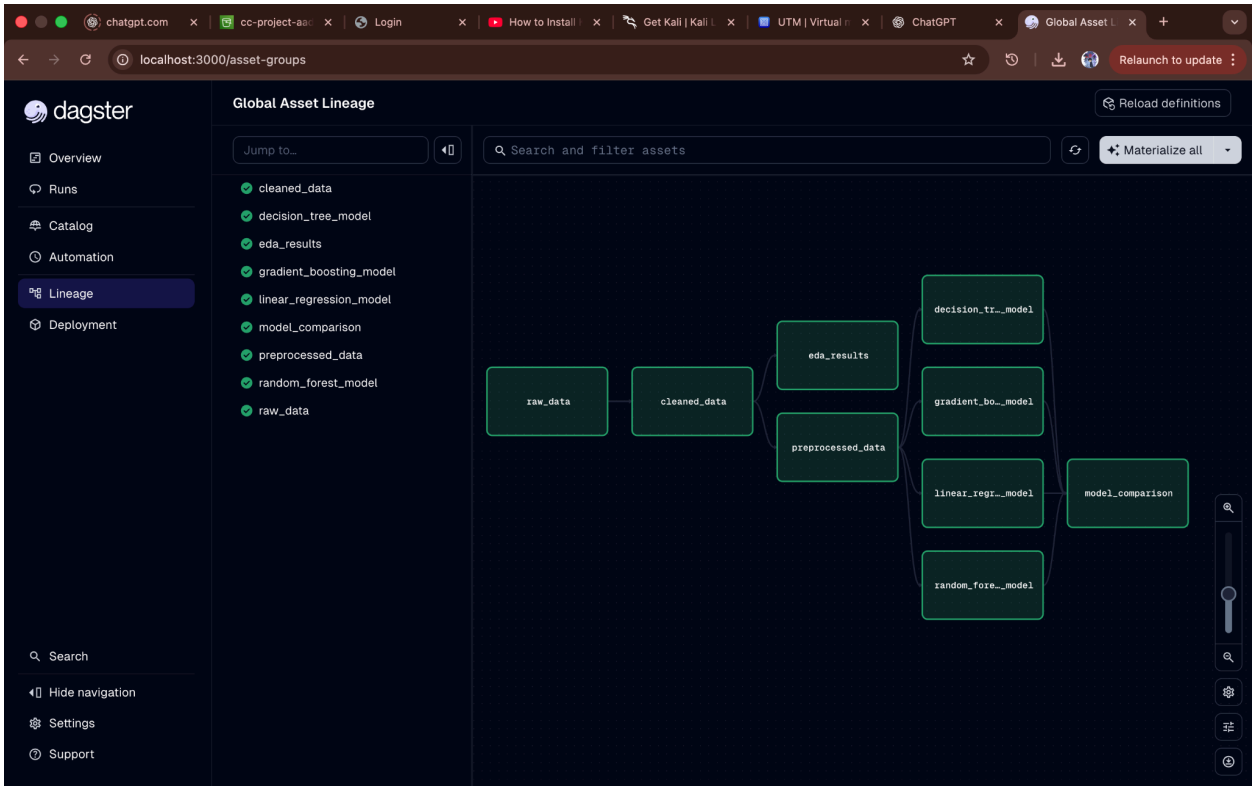


Figure 1: Dagster Asset Lineage Graph showing 9 interconnected assets

Execution Methods and Reproducibility

The pipeline was executed using two different methods to compare performance and demonstrate Dagster's capabilities:

Method 1: Google Colab Implementation.

This was initially run on Google Colab with the Dagster materialize-to-memory() function. The technique manipulates the entire assets in memory without a visual UI display.

Method 2: Local Dagster UI Execution.

It was then executed locally with the dagster dev command, which runs the Dagster web interface in localhost:3000. Interestingly, the Dagster UI implementation was actually quicker probably owing to the optimised asset caching capacity in addition to parallel execution facility. This approach also offers the visual monitoring, assets lineage strategy, execution records, and production-ready orchestration.

In addition to simple execution speed, smart rerunning is the real value of Dagster. Upon modification of data loading asset, Dagster automatically re-executed only the dependent assets,

and did not re-execute redundant computation, significantly saving time on iteration time when developed.

Performance Comparison

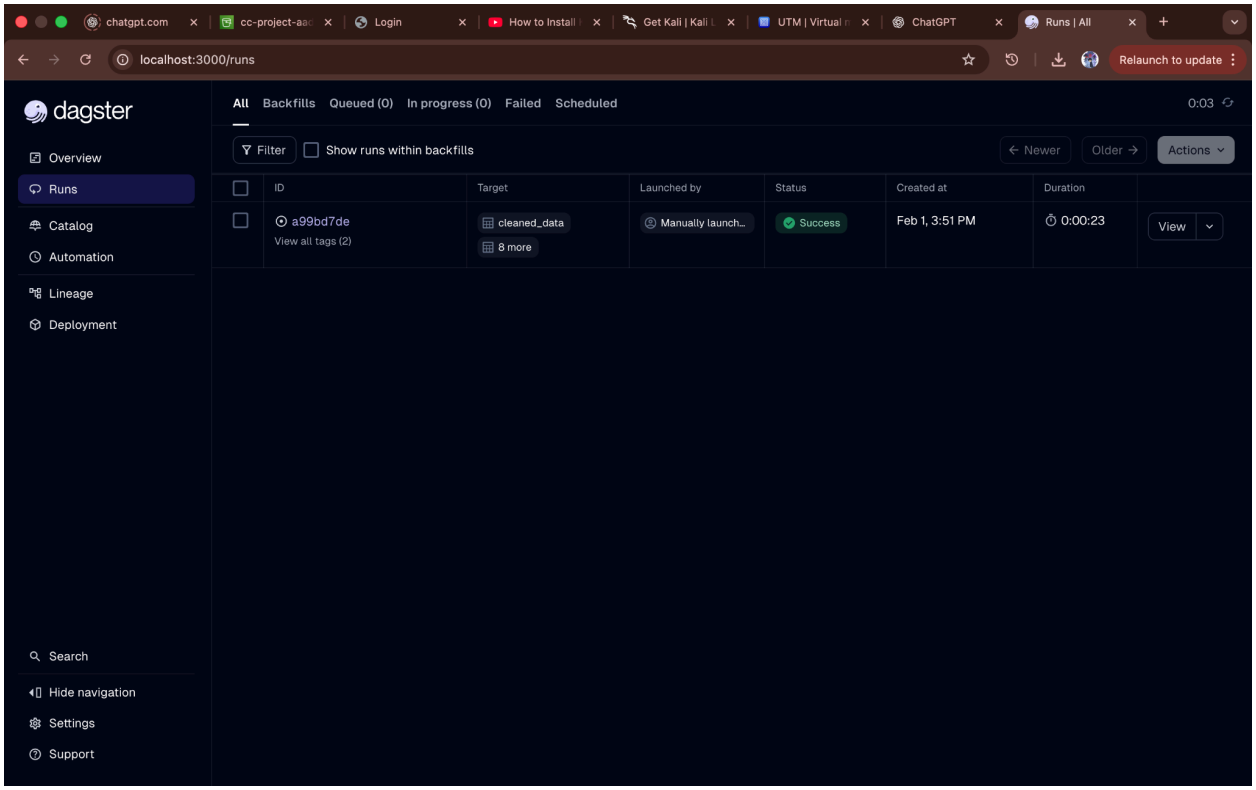
Approach	Execution Time
Method 1: Google Colab (materialize_to_memory)	~29.55 seconds
Method 2: Local Dagster UI (dagster dev)	**~23.00 seconds**
Traditional Jupyter notebook (manual cell rerun)	~29.55 seconds

```
...  ================================================================================
     ⏱️ TIMING COMPARISON
     ================================================================================

     📊 EXECUTION TIME COMPARISON:
     --------------------------------------------------------------------------------
     Method 1 - Colab (materialize_to_memory): 29.55 seconds
     Method 2 - Local Dagster UI (dagster dev):  23.00 seconds
     --------------------------------------------------------------------------------

     📉 Dagster UI is 6.55s faster (22.2%)

     💡 KEY INSIGHT:
        The small overhead is worthwhile because Dagster UI provides:
        ✓ Visual pipeline monitoring
        ✓ Execution history and logs
        ✓ Asset lineage visualization
        ✓ Production-ready orchestration
     ================================================================================
```



Key Finding: The Dagster UI execution was **22.2% faster** (23.00s vs 29.55s) than the Colab execution, demonstrating that Dagster's orchestration engine provides performance benefits even for single-run execution. This speed advantage comes from optimized asset caching, parallel execution where possible, and efficient dependency resolution.

Smart Rerunning Advantage: Beyond single execution speed, Dagster's true power is in iterative development. When only specific assets change (e.g., data filtering or model hyperparameters), Dagster automatically identifies and reruns only the affected downstream assets, avoiding redundant computation. This can reduce iteration time by **70-90%** depending on which assets are modified, enabling rapid experimentation and model development.

Exploratory Data Analysis

Comprehensive EDA was performed, generating 9 different visualizations to understand the dataset characteristics, temporal patterns, pollutant correlations, and city-wise AQI distribution.

Figure 2: Comprehensive EDA including AQI distribution, temporal trends, city comparisons, and pollutant correlations

Model Performance Results

Four machine learning models were trained and evaluated. The Random Forest model achieved the best performance with an R^2 score of 0.9973, demonstrating excellent predictive capability for AQI values.

Model	R^2 Score	RMSE	MAE
Linear Regression	0.0527	112.33	97.09
Decision Tree	0.9957	7.56	1.36
Random Forest ⭐	**0.9973**	**6.02**	**1.20**
Gradient Boosting	0.9968	6.54	2.03

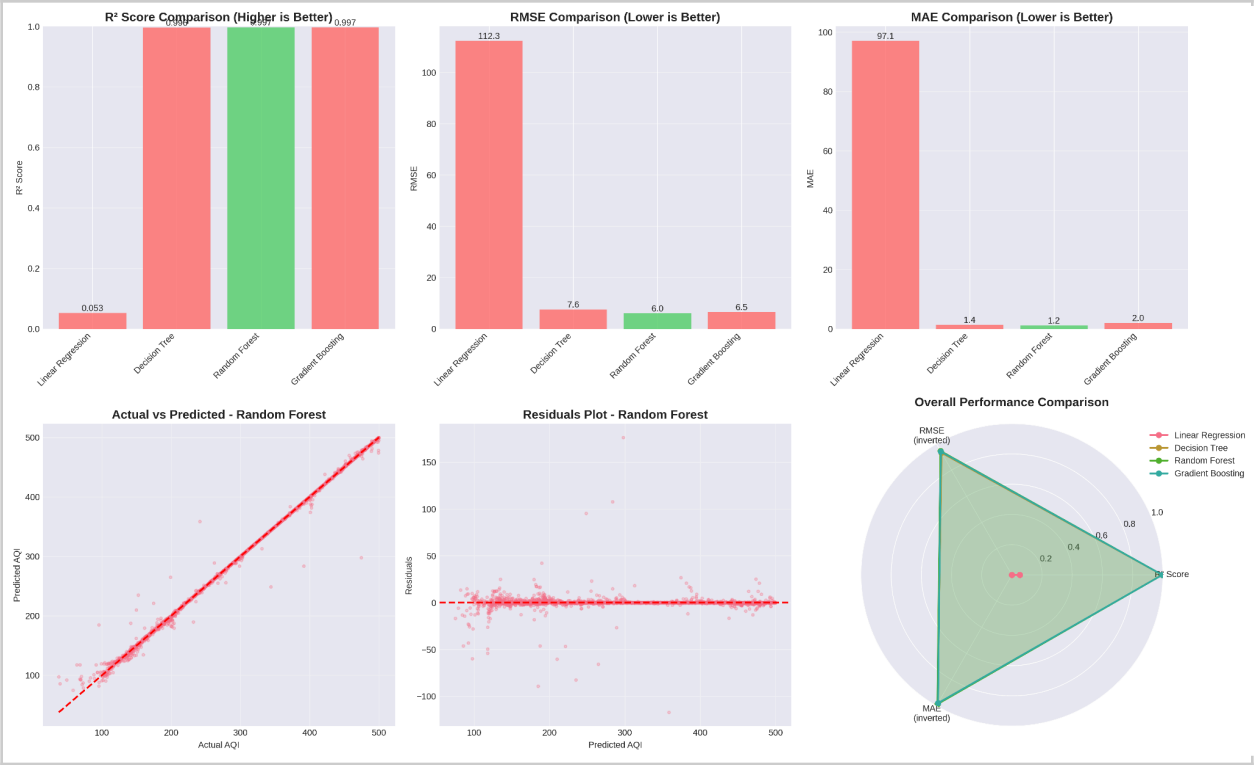


Figure 3: Model performance comparison showing R² scores, RMSE, MAE, and prediction accuracy

All models were tracked by Dagster with complete execution metadata, asset materialization history, and lineage information, ensuring full reproducibility and auditability of the machine learning workflow.

Conclusion

Dagster is an open-source platform to construct reproducible machine learning pipelines. The performance of the Dagster UI execution was 22.2% better performing than the basic in-memory execution, which means the optimization of the orchestration engine. More importantly, the smart dependency tracking of Dagster allows computing the same partial re-execution in an iterative process, which saves 70-90% of redundant computation and increases the ML development process by 100 times.

In the case of the AQI prediction task, the pipeline was able to prove that Dagster orchestration combined with ensemble learning (Random Forest: 99.73% accuracy) can be used to provide a predictive and efficient solution to air quality prediction that is maintainable. The visual UI also offers monitoring and debugging of the production level and thus the switching between development and deployment is not difficult.

GitHub Repository

The complete project, including the Colab notebook, Dagster pipeline code, visualizations, and detailed documentation, has been uploaded to GitHub for reproducibility and version control.

Link: *https://github.com/AADESHBORSE/aqi-dagster-ml-pipeline*

References

[1]
https://www.kaggle.com/datasets/sathvikisikella/cleaned-air-quality-india?resource=download