Nicole Gallo
8/19/2025
D602 - QBN1 Task 2: Data Production Pipeline

Airport list:

- ATL: Atlanta, Georgia
- LAX: Los Angeles, California
- JFK: New York, New York
- MIA: Miami, Florida
- DFW: Dallas, Texas
- **ORD: Chicago, Illinois**

A. Create your subgroup and project in GitLab using the provided web link and the "GitLab How-To" web link by doing the following:
    a. Clone the project to the IDE.
    b. Commit with a message and push when you complete each requirement listed in parts B, C, D, and E.

    Note: You may commit and push whenever you want to back up your changes, even if a requirement is not yet complete.

    c. Submit a copy of the GitLab repository URL in the "Comments to Evaluator" section when you submit this assessment.
    d. Submit a copy of the repository branch history retrieved from your repository, which must include the commit messages and dates.

B. Write a script in either Python or R to import the data you downloaded and format it according to the criteria required by the model script, demonstrating a progression of work on your code. You must run a DVC command to create a metafile for your dataset and submit the metafile to the GitLab repository. Submit *at least* **two** versions of your code to the GitLab repository demonstrating a progression of work on your code.

    Note: Include the original CSV file you downloaded in your GitLab repository. Although data files are not typically stored in GitLab repositories, including this file in your submission is necessary to allow evaluators to fully assess your work.

C. Write a script in either Python or R to filter data to only departures from the chosen airport, then implement *at least* **two** other data cleaning steps. Submit *at least* **two** versions of your code to the GitLab repository demonstrating a progression of work on your code.

D. Using the code template provided in the GitLab repository, implement an MLFlow experiment that captures the features listed in the comments within the poly_regressor file for either Python or R. Submit *at least* **two** versions of your code to the GitLab repository demonstrating a progression of work on your code.

E. Using the provided YAML file for either Python or R, write an MLProject file that links the two scripts you wrote in parts B and C with the modified poly_regressor script from part D.

    Note: Your MLProject file must be submitted to the GitLab repository.

F. **Provide a written explanation of how you wrote the code and MLProject pipeline, including any challenges you encountered and how you addressed these challenges. Include a screenshot of your MLProject pipeline running successfully.**

I implemented a small, reproducible machine learning pipeline using Python that imports Bureau of Transportation Statistics (BTS) On-Time Performance data, filters and cleans it for a single departure airport, and trains a polynomial regression model while tracking experiments in MLflow. I used GitLab for version control, DVC for dataset tracking (metafile), and an MLflow MLProject file to link the steps into one end-to-end run.

Repository structure:

```
configs/                        # schema configs
data/
    raw/                        # original BTS CSV (and a DVC-tracked ZIP copy)
    processed/                  # cleaned_data.csv (final model input)
docs/                           # branch history export and notes
logs/                           # step logs (import, clean, regression)
scripts/                        # scripts for Parts B, C
MLproject                       # MLflow project spec
pipeline_env.yaml               # Conda environment
poly_regressor_Python_1.0.0.py  # Part D regression model
```

Step 1: Import and Format

- **Goal:** read the raw BTS CSV/ZIP and format it for later steps.
- **Version 1 (import_and_format_v1.py):** loads the raw CSV, prints shape/columns, and exports a small sample.
- **Version 2 (import_and_format_v2.py):** validates schema, standardizes column names, normalizes HHMM time fields, drops critical nulls, and writes `data/processed/cleaned_data.csv`.
- **DVC metafile:** Because the original CSV must remain in Git for evaluation, I DVC-tracked a ZIP copy (`data/raw/T_ONTIME_REPORTING.zip`) and committed the generated `.dvc` metafile. The CSV stays in Git; the ZIP is tracked by DVC.

Step 2: Filter and clean

- **Goal:** keep only departures from my chosen airport (ORD) and perform ≥ 2 additional cleaning steps.
- **Version 1 (filter_airport_v1.py):** filters to origin airport, drops cancelled/diverted flights, and writes `cleaned_data.csv`.
- **Version 2 (filter_airport_v2.py):** expands cleaning:
- Converts scheduled HHMM times into datetime, then derives features like weekday and hour of day.
- Removes implausible delays (e.g., > 60 minutes) and drops duplicates.
- Produces a tidy modeling set with columns needed for the regression step.

Step 3: Polynominal Regression with MLflow

- **Goal:** train and evaluate a regression model while logging experiments.
- **Version 1 (poly_regressor_Python_1.0.0.py, baseline):**
  - Trains Ridge regression with polynomial features.
  - Logs parameters and metrics for a single run.
- **Version 2 (poly_regressor_Python_1.0.0.py, updated):**
  - Implements an **alpha sweep** using `num_alphas` parameter.
  - Uses nested MLflow runs: parent run logs setup (polynomial order, bounds); child runs log per-alpha metrics.
  - Logs artifacts:
    - `logs/polynomial_regression.txt` (training log)
    - `model_performance_test.jpg` (predicted vs. actual plot)
    - `finalized_model.pkl` (best Ridge model)
    - `airport_encodings.json` (one-hot encoding map)
- Fixed the **run ID conflict bug** by hard-coding the experiment name `"Airport Departure Delays"` and passing it consistently via CLI.

Step 4: MLProject pipeline
I created two files in the repo root to run the pipeline:

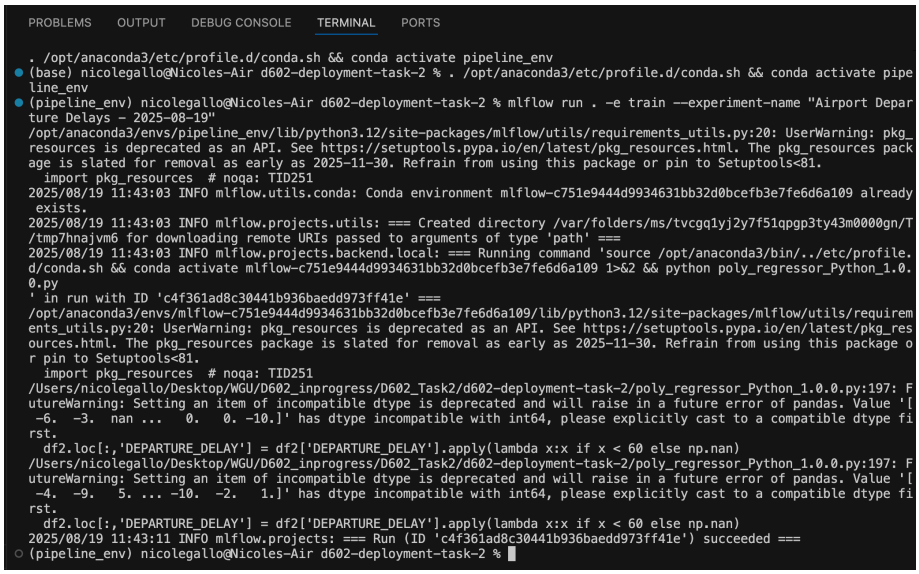
- **pipeline_env.yaml**: Conda env spec including pandas, numpy, scikit-learn, seaborn, matplotlib, and mlflow.
- **MLproject**: defines entry points for each step:
  - `import_format` → import_and_format_v2.py
  - `filter_airport` → filter_airport_v2.py
  - `train` → poly_regressor_Python_1.0.0.py (with `--num-alphas` param)
  - `pipeline` → runs B → C → D in sequence

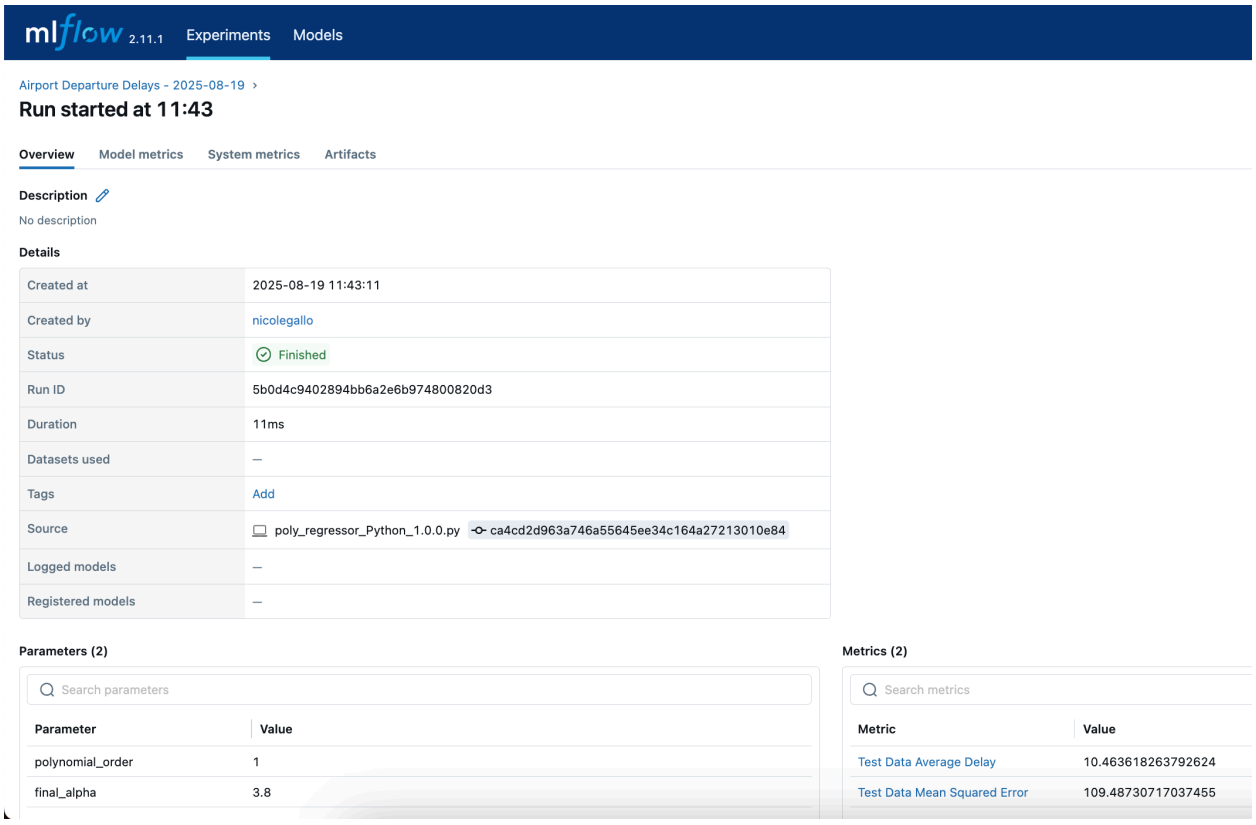
I executed the full pipeline using the following command:

```
mlflow run . -e train --experiment-name "Airport Departure Delays" -P num-alphas=20
```

Challenges & Fixes

1. **DVC vs Git conflict for the raw dataset**
    a. *Issue:* `dvc add` refused to track `data/raw/T_ONTIME_REPORTING.csv` because it was already tracked by Git.
    b. *Fix:* To satisfy the course requirement (original CSV must remain in Git), I DVC-tracked a **ZIP copy** instead (`data/raw/T_ONTIME_REPORTING.zip`) and committed the `.dvc` metafile. The CSV stays in Git; the ZIP is ignored by Git but tracked by DVC.
2. **Python environment errors (e.g., `ModuleNotFoundError: pandas`)**
    a. *Issue:* Running scripts with a Python interpreter that didn't have the required packages.
    b. *Fix:* Created/activated a project virtualenv (`.venv`), installed dependencies (`pandas, numpy, scikit-learn, mlflow, matplotlib, pyyaml`), and ran all scripts with that interpreter.
3. **Experiment name conflict**
    a. *Issue:* Runs failed with `active run ID does not match environment run ID`.
    b. *Fix:* Changed to a fixed experiment name "`Airport Departure Delays`" to avoid run ID mismatch errors.
4. **Numerical stability warnings**
    a. *Issue*: Ridge regression sometimes triggered `LinAlgWarning: Ill-conditioned matrix`.
    b. *Fix*: Restricted delays > 60 min during cleaning, used Ridge regularization, and logged metrics (`MSE`, √`MSE`) for interpretability.
5. **MLflow UI port conflict**
    a. *Issue*: Running `mlflow ui` sometimes failed with "Connection in use: ('127.0.0.1', 5000)".
    b. *Fix*: Launched the UI on an alternate port using `mlflow ui --port 5001`.





G. **Sources**: No external resources were used beyond the WGU course resources, instructors' course tips, and the corresponding webinars for D602.