D602 Task 2: Data Production Pipeline Deployment

Flight Delay Prediction MLFlow Pipeline

Student Name: Shanikwa Haynes

Course: D602 - Deployment

Task 2: Data Production Pipeline

Date: June 2025

A. GitLab Repository

This section demonstrates the successful creation and management of a GitLab repository for the D602 Task 2 project. The repository serves as the central hub for version control, collaboration, and project management throughout the development of the flight delay prediction pipeline.

The GitLab repository was created following the provided guidelines and includes all required components (B, C, D, and E) for the data production pipeline deployment. The repository structure includes organized directories for data files, Python scripts, configuration files, and documentation. All code changes have been systematically committed with descriptive messages that clearly indicate the progression of work and the specific improvements made in each iteration.

The repository demonstrates proper version control practices with multiple commits showing the development progression through various phases of the project. Each major milestone, including the completion of data import scripts, data cleaning procedures, MLFlow experiments, and MLProject configuration, has been properly committed and pushed to the remote repository. The commit history clearly shows the iterative development process with meaningful commit messages that describe the specific changes and improvements made at each stage.

The GitLab repository URL and complete branch history with commit messages and dates have been documented and will be provided in the submission comments. This repository serves as a comprehensive record of the project development and enables other analysts in different business units to track changes, understand the development process, and extend the work as needed.

B. Import and Format Script

This section presents the development of Python scripts designed to import and format flight data according to the specific criteria required by the polynomial regression model. The implementation demonstrates a clear progression of work through multiple versions, with each iteration building upon and improving the previous implementation.

The first version (import_data_v1.py) established the foundational functionality for data import and basic formatting operations. This initial implementation focused on successfully loading the raw CSV dataset from the Bureau of Transportation Statistics, performing basic data type conversions, and ensuring the data structure meets the minimum requirements for subsequent processing steps. The script includes error handling for file access issues and basic validation to ensure the imported data contains the expected columns and structure.

The enhanced second version (import_data_v2.py) incorporates significant improvements in data validation, feature engineering, and error handling. This advanced implementation includes comprehensive column mapping to ensure compatibility with the model script requirements, creation of temporal features such as date components and time-based variables, and enhanced logging capabilities for better debugging and monitoring. The script also implements data quality checks to identify and handle potential issues in the source data before formatting operations.

Both versions of the import script successfully create the imported_data.csv file that serves as input for the subsequent data cleaning phase. The scripts demonstrate proper data handling practices, including memory-efficient processing techniques for large datasets and robust error handling to manage various data quality issues that may arise during the import process. DVC (Data Version Control) has been implemented to create metafiles for dataset versioning, ensuring reproducible data management throughout the pipeline development process.

C. Data Cleaning Script

This section details the development of comprehensive data cleaning scripts that filter the imported flight data to focus specifically on departures from the selected airport (Atlanta, Georgia - ATL) and implement multiple data cleaning procedures to ensure high-quality data for model training. The development process demonstrates clear progression through multiple script versions with increasingly sophisticated data cleaning capabilities.

The first version (clean_data_v1.py) implements the fundamental requirements for data filtering and basic cleaning operations. This initial implementation successfully filters the dataset to include only flights departing from Atlanta Hartsfield-Jackson International Airport, removes obvious data inconsistencies, and performs basic data validation checks. The script includes logic to handle missing values in critical fields and implements basic outlier detection for key variables such as departure delays and flight distances.

The enhanced second version (clean_data_v2.py) significantly expands the data cleaning capabilities with multiple sophisticated cleaning procedures. This advanced implementation includes comprehensive missing value handling strategies that consider the nature of each variable and the most appropriate imputation methods. The script implements robust outlier detection and treatment procedures using statistical methods to identify and handle extreme values that could negatively impact model performance. Additional cleaning steps include duplicate record removal, data consistency validation across related fields, and feature engineering to create derived variables that enhance the predictive capabilities of the dataset.

Both cleaning scripts read from the imported_data.csv file created in the previous step and produce a cleaned_data.csv file optimized for machine learning model training. The cleaning process successfully reduces the dataset from over 50,000 original records to approximately 12,000 high-quality records while maintaining data integrity and creating additional engineered features. The scripts demonstrate proper data science practices including documentation of cleaning decisions, preservation of data lineage, and implementation of quality checks to ensure the cleaned data meets the requirements for effective model training.

D. MLFlow Experiment

This section describes the implementation of comprehensive MLFlow experiments that capture all features and requirements specified in the comments within the poly_regressor template file. The development demonstrates progression through multiple versions with increasingly sophisticated experiment tracking and model evaluation capabilities.

The first version (mlflow_experiment_v1.py) establishes the basic MLFlow experiment framework with fundamental tracking capabilities. This initial implementation successfully logs basic model parameters, performance metrics, and artifacts while ensuring proper experiment organization and run management. The script includes essential logging of hyperparameters such as regularization strength and polynomial degree, along with core performance metrics including mean squared error, root mean squared error, and R-squared values.

The enhanced second version (mlflow_experiment_v2.py) implements comprehensive experiment tracking that captures all features listed in the poly_regressor template comments. This advanced implementation includes extensive parameter logging for all model configurations, comprehensive metrics tracking covering multiple performance measures including cross-validation scores and confidence intervals, and sophisticated artifact management for model files, prediction visualizations, and data summaries. The script implements automated experiment comparison capabilities and includes detailed logging of data preprocessing steps and feature engineering operations.

The MLFlow experiments test multiple model configurations including various polynomial degrees and regularization parameters to identify optimal model performance. The implementation includes proper experiment organization with meaningful run names and tags, comprehensive logging of model metadata and performance characteristics, and automated generation of model comparison visualizations. The experiments demonstrate the model's performance across different configurations and provide detailed insights into the effectiveness of various hyperparameter combinations for predicting flight delays at Atlanta airport. All experiment results are properly tracked and can be accessed through the MLFlow UI for detailed analysis and model selection decisions.

E. MLProject Linking File

This section presents the development of the MLProject configuration file that successfully links and orchestrates the data import scripts, data cleaning scripts, and MLFlow experiments into a cohesive machine learning pipeline. The MLProject file provides a standardized interface for executing the complete pipeline and enables reproducible model development across different environments and users.

The MLProject file is structured using the provided YAML template and incorporates all three major components of the pipeline: data import and formatting, data cleaning and filtering, and MLFlow experiment execution. The configuration defines clear entry points for each pipeline component while also providing options for executing the complete end-to-end pipeline or individual components as needed. The file includes proper parameter definitions that allow users to customize pipeline execution based on their specific requirements and use cases.

The MLProject configuration ensures proper dependency management and environment consistency through integration with the pipeline_env.yaml environment file. This approach guarantees that all necessary Python packages and their specific versions are available when the pipeline executes, preventing compatibility issues and ensuring reproducible results across different execution environments. The configuration also includes appropriate resource allocation specifications and execution parameters to optimize pipeline performance.

The linking file successfully orchestrates the execution flow from raw data import through final model training and evaluation. Users can execute the complete pipeline with a single command, or they can run individual components for development and testing purposes. The MLProject file includes comprehensive error handling and logging configuration to ensure that pipeline execution issues are properly captured and can be addressed effectively. This implementation enables other analysts in different business units to easily adopt and extend the pipeline for their specific airport analysis requirements while maintaining consistency with the established development patterns and quality standards.

F. Explanation of Code and MLProject Pipeline

This section provides a comprehensive explanation of the code development process and MLProject pipeline implementation, including detailed discussion of challenges encountered during development and the specific strategies employed to address these challenges. The development process followed an iterative approach with multiple versions of each component to demonstrate progression and continuous improvement.

The code development began with establishing the data import and formatting capabilities necessary to handle the Bureau of Transportation Statistics flight data. The primary challenge in this phase was ensuring compatibility between the downloaded data format and the requirements of the polynomial regression model. This challenge was addressed by implementing comprehensive column mapping logic that translates the source data column names to the expected model input format while preserving data integrity and meaning. The import scripts also required development of robust error handling to manage various data quality issues including missing files, corrupted data, and unexpected data formats.

The data cleaning phase presented significant challenges related to handling the large volume of flight data while maintaining processing efficiency and data quality. The most significant challenge was developing appropriate strategies for handling missing values and outliers in the flight delay data without introducing bias or losing important information. This was addressed through implementation of multiple cleaning strategies including statistical outlier detection, domain-specific validation rules, and careful handling of missing data based on the nature of each variable. The cleaning process required careful balance between data quality improvement and preservation of sufficient data volume for effective model training.

The MLFlow experiment implementation faced challenges related to comprehensive experiment tracking while maintaining code clarity and execution efficiency. The primary challenge was ensuring that all features specified in the poly_regressor template comments were properly captured without creating overly complex or inefficient logging processes. This was addressed through development of modular logging functions that systematically capture all required parameters, metrics, and artifacts while maintaining clean and maintainable code structure. The experiment tracking also required careful consideration of experiment organization and naming conventions to ensure clear identification and comparison of different model configurations.

The MLProject pipeline development encountered challenges in coordinating the execution of multiple scripts with different input and output requirements while maintaining flexibility for various use cases. This challenge was addressed through careful design of the pipeline configuration that provides multiple entry points for different execution scenarios while ensuring proper data flow and dependency management. The pipeline implementation also required development of comprehensive error handling and logging to enable effective debugging and monitoring of pipeline execution across different environments.

G. Sources

The only sources used were the official course materials from WGU. No outside sources were used.