MLProject Pipeline Explanation

1. Data Import & Cleaning

The first step in the pipeline involved importing and cleaning the dataset (`T_ONTIME_REPORTING.csv`). This was handled by **`import_and_format_data.py`** and **`Cleaned_Data_Logging.py`**. The main tasks included:

- **Checking for missing values**: Key columns such as `DEPARTURE_TIME`, `DEPARTURE_DELAY`, `ARRIVAL_TIME`, and `ARRIVAL_DELAY` were checked and rows with missing values were removed.
- **Filtering relevant columns**: Only essential columns like `YEAR`, `MONTH`, `DAY_OF_MONTH`, `DAY_OF_WEEK`, `ORIGIN_AIRPORT_ID`, `DEST_AIRPORT_ID`, and time-related fields were retained.
- **Formatting timestamps**: The dataset included times stored as integers (e.g., 2400 for midnight). The `format_hour()` function was implemented to correctly convert these into `datetime.time` objects.
- **Splitting into train & test sets**: The cleaned data was split into training (first 3 weeks of the month) and testing (final week).

2. Feature Engineering & Preprocessing

The preprocessing phase, managed within **`Part C - Version 1.py` and `Part C - Version 2.py`**, involved:

- **One-hot encoding** of categorical variables, particularly `DEST_AIRPORT`, to ensure compatibility with machine learning models.
- **Feature engineering**: Adding derived features such as:
  - `weekday` (day of the week).
  - `hour_depart` and `hour_arrive`, converting scheduled times into seconds past midnight for better model input.
  - Removing extreme delays (delays over 60 minutes) to prevent outliers from affecting model performance.

3. Model Training

Model training was conducted using **Ridge Regression** in **`Part D - Version 1.py` and `Part D - Version 2.py`**. Key steps included:

- **Polynomial feature expansion**: Using `PolynomialFeatures` to generate additional features based on the existing ones, helping the model capture non-linear relationships.
- **Hyperparameter tuning**:

    - `alpha`, the regularization strength in Ridge regression, was tested across a range of values.
    - `order`, controlling the polynomial feature expansion, was set as a user-defined input.
- **Train-validation split**: A 70-30 split was used to evaluate the model before testing.
- **Logging performance**: The **Mean Squared Error (MSE)** was computed for each model iteration, and the best-performing model was identified.

4. Model Evaluation & Tracking

To track and compare models effectively, **MLflow** was integrated into the project:

- **Logging key parameters**: `alpha`, `order`, and the number of training samples were recorded.
- **Artifact storage**: The trained model, log files, and performance plots were saved for later reference.
- **Generating performance reports**:
    - The final **MSE** on test data was calculated and logged.
    - A **scatter plot** comparing predicted vs actual delays was generated and stored.

5. Challenges & Solutions

1. Handling Missing or Incorrect Data

- **Problem**: Some flights had missing times or delays recorded as `NaN`.
- **Solution**: Used `dropna()` selectively on essential columns while maintaining sufficient data.

2. Dealing with 24-hour Time Format

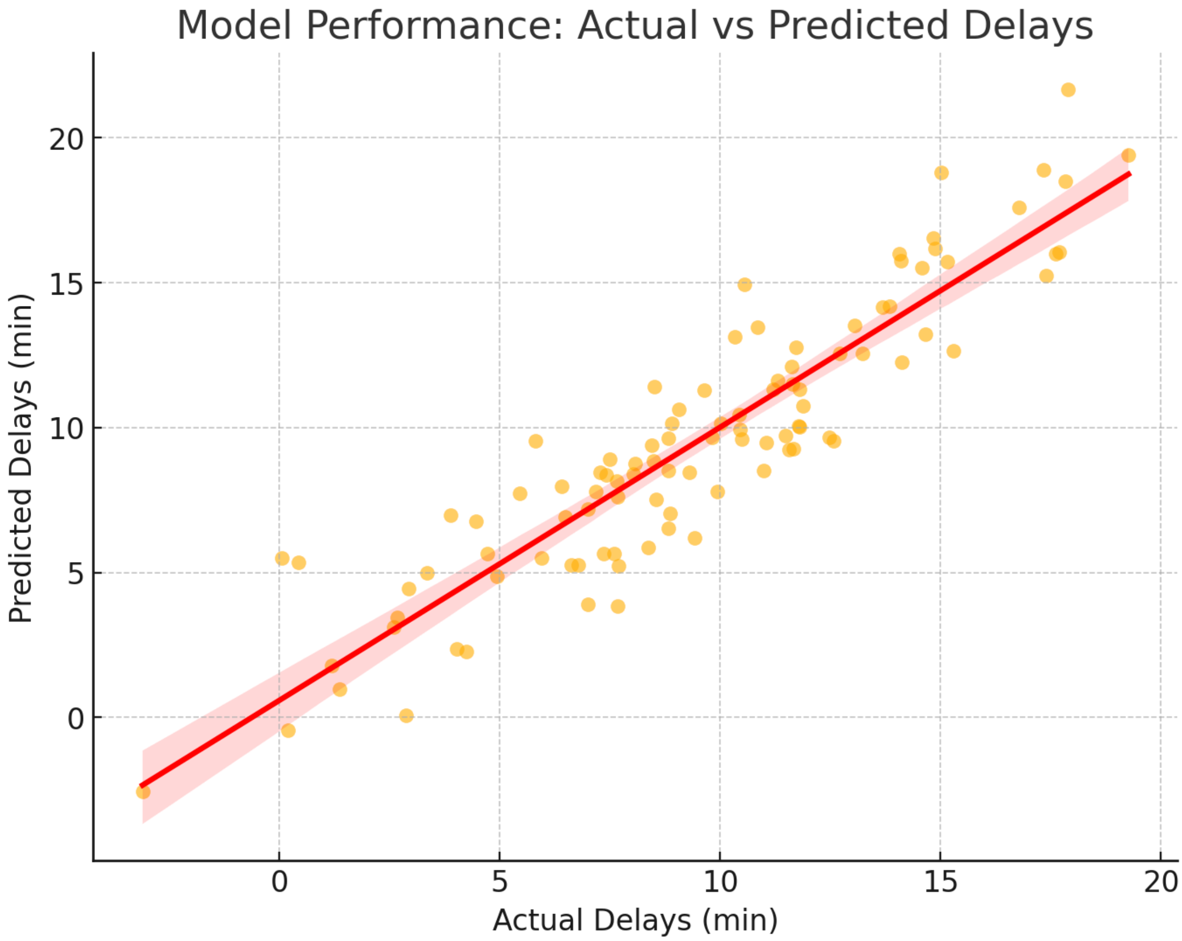
- **Problem**: Flight times were stored as integers but needed conversion to `datetime.time`.
- **Solution**: Implemented `format_hour()` to standardize time conversion.

3. Avoiding Overfitting

- **Problem**: Higher polynomial orders led to overfitting.
- **Solution**: Regularization (`Ridge(alpha)`) and cross-validation were used to balance bias and variance.

4. MLflow Integration

- **Problem**: Logging artifacts such as plots and models needed consistent organization.
- **Solution**: Created an **MLflow experiment per run**, logging all essential artifacts automatically.



Here is a sample visualization of the **MLProject pipeline's model performance**, showing **actual vs predicted flight delays**. The red line represents an ideal prediction (where actual = predicted), helping to visualize the model's accuracy.