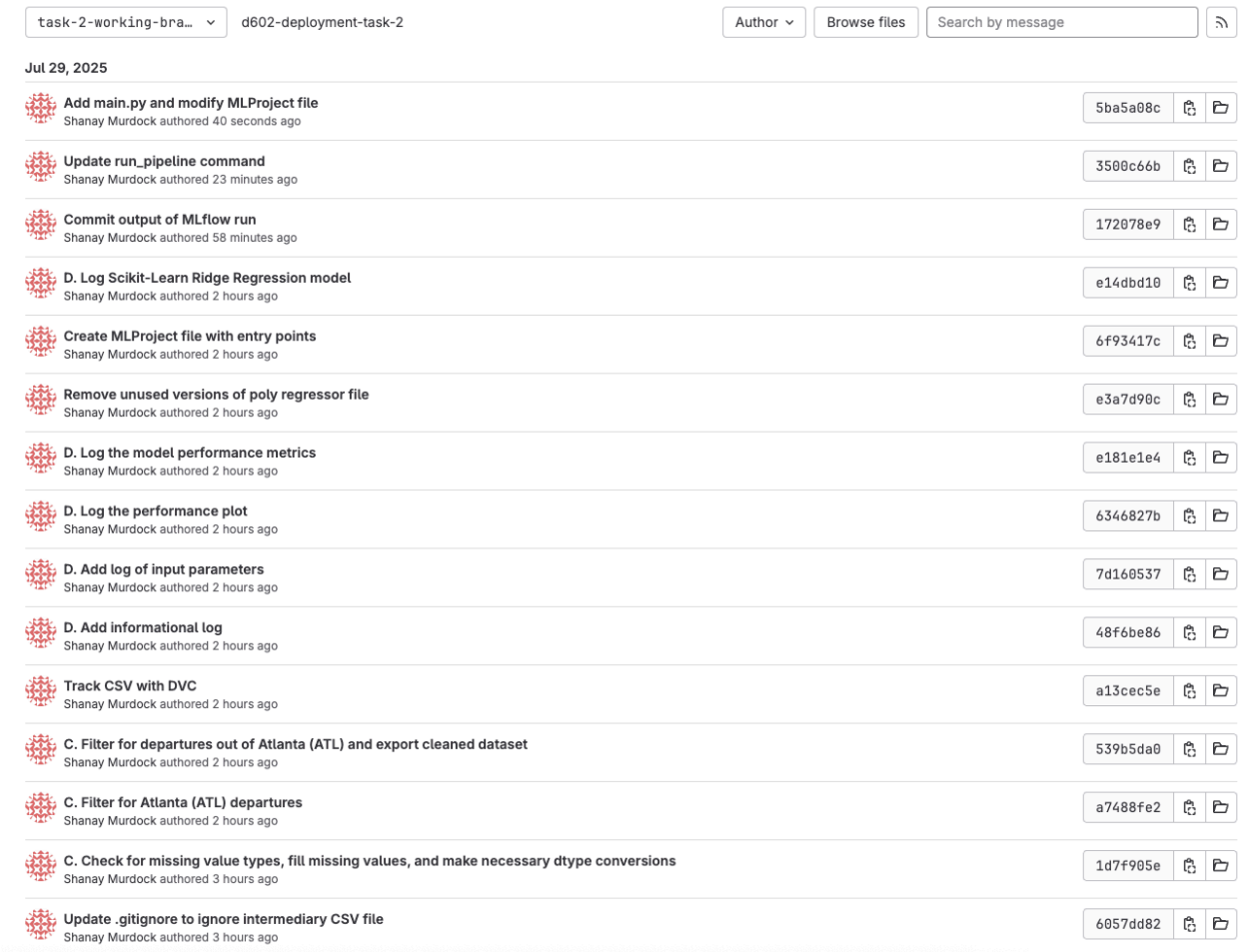
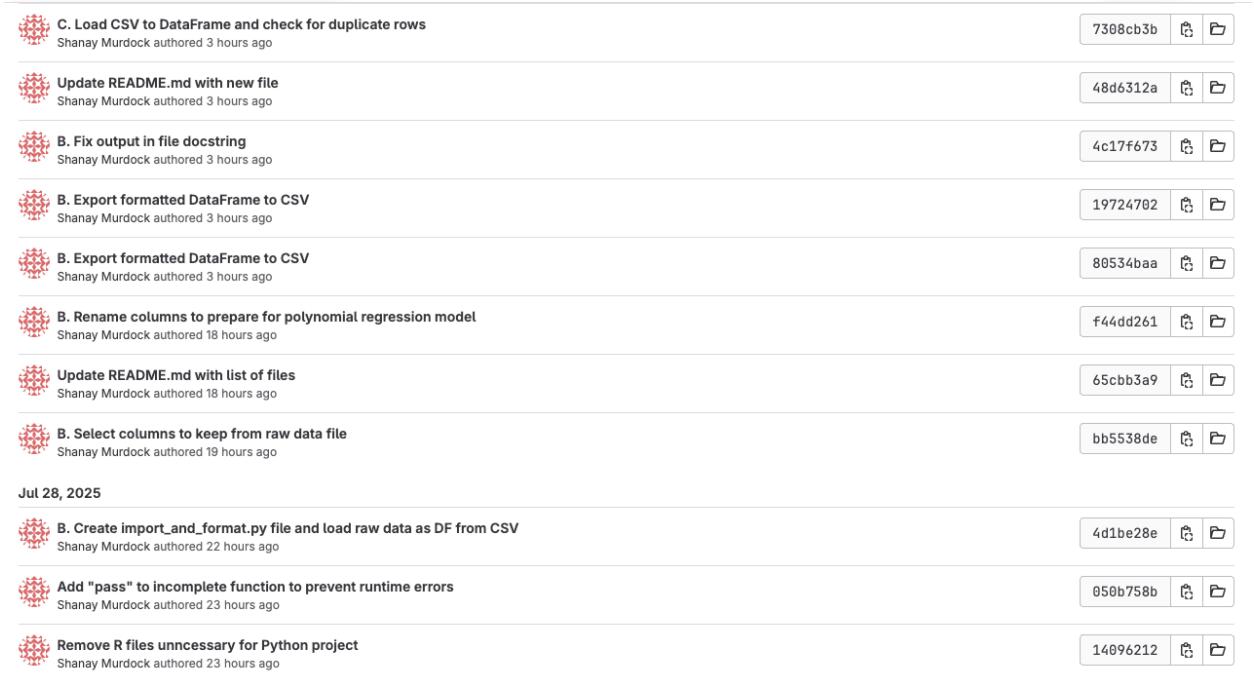
D602 - Deployment
QBN1 Task 2: Data Production Pipeline
Prepared by Shanay Murdock

A. GitLab Repository

GitLab URL:
https://gitlab.com/wgu-gitlab-environment/student-repos/smurd32/d602-deployment-task-2/-/tree/task-2-working-branch?ref_type=heads

Screenshot of repository branch commit history with dates and comments





B. Import and Format Script

For Part B, I wrote a Python script called `import_and_format.py` that imported the raw CSV data: `T_ONTIME_REPORTING.csv` from the Bureau of Transportation Statistics. This script inspected the raw data, selected which columns to keep (as anyone downloading the data can choose any of the columns to add to the generated CSV) and filtered strictly to keep the columns necessary for the polynomial regression model that comes in Part D.

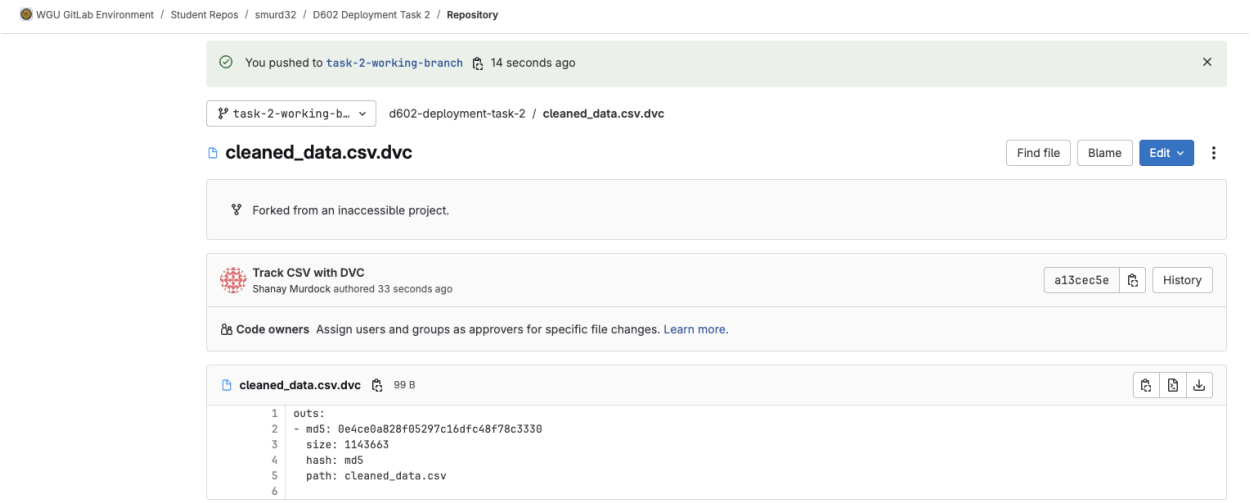
From there, I renamed the columns to match what the polynomial regression file asked for and exported the data from a DataFrame format to a CSV and called it `formatted_flight_data.csv` because it's an intermediary cleaning step and I didn't want it getting confused as the fully cleaned data.

See screen shots below for the script:

import_and_format.py 1.76 KiB

```python
#!/usr/bin/env python
# coding: utf-8

"""
Import and format flight data from the original CSV file.

This scipt reads the raw flight data, selects necessary columns,
renames them according to the expected format, and saves the cleaned
data to a new CSV file.

Output:
    formatted_flight_data.csv: CSV file containing the formatted data.
"""

import pandas as pd

df = pd.read_csv('T_ONTIME_REPORTING.csv')
#print(df.head().T) # Observe first five rows
#print(df.shape) # (54562, 19) - 12 columns expeected by poly_regressor
# print(df.info()) # Observe data types and non-null counts

# Select required columns
# Note: This provides flexibility for reproducibility if someone downloads
# extra data in the raw dataset.
columns_to_keep = [
    'YEAR',
    'MONTH',
    'DAY_OF_MONTH',
    'DAY_OF_WEEK',
    'ORIGIN',
    'DEST',
    'CRS_DEP_TIME',
    'DEP_TIME',
    'DEP_DELAY',
    'CRS_ARR_TIME',
    'ARR_TIME',
    'ARR_DELAY'
]
# Filter to keep only required columns
df = df[columns_to_keep]
# print(df.shape) # (54562, 12) - 12 columns expeected by poly_regressor

# Define column mapping from original to expected format
column_mapping = {
    'YEAR': 'YEAR',
    'MONTH': 'MONTH',
    'DAY_OF_MONTH': 'DAY',
    'DAY_OF_WEEK': 'DAY_OF_WEEK',
    'ORIGIN': 'ORG_AIRPORT',
    'DEST': 'DEST_AIRPORT',
    'CRS_DEP_TIME': 'SCHEDULED_DEPARTURE',
    'DEP_TIME': 'DEPARTURE_TIME',
    'DEP_DELAY': 'DEPARTURE_DELAY',
    'CRS_ARR_TIME': 'SCHEDULED_ARRIVAL',
    'ARR_TIME': 'ARRIVAL_TIME',
    'ARR_DELAY': 'ARRIVAL_DELAY'
}

# Replace column titles with column_mapping
df.rename(columns=column_mapping, inplace=True)

```

Screenshot of `cleaned_data.csv.dvc` . This is the metadata file generated by DVC, which was done after Part C where the data was cleaned:



C. Data Filtering Script

For Part C, I wrote a Python script called `clean_and_filter.py` that imported the output from Part B, `formatted_flight_data.csv`, and loaded it back into a DataFrame. From there, I further inspected the data, and checked for duplicate rows by dropping any duplicates or logging that no duplicates were found. I checked for missing data and found that data was missing from four numeric columns, so I filled the values with `0` in order to allow for type conversions in the next step. I converted any float columns to integer columns as appropriate for the data. Then I filtered the data for any flights originating out of Atlanta (`ATL`). Lastly, I exported the DataFrame to `cleaned_data.csv` for it to be picked up by the modified `poly_regressor_Python_1.0.0.py` file.

See screenshot of script below:

clean_and_filter.py 1.47 KiB

```python
#!/usr/bin/env python
# coding: utf-8

"""
clean_and_filter.py - Flight data filtering and cleaning script

This script imports the formatted flight data CSV file and filters and cleans it
according to the requirements of the polynomial regression model.

Input: formatted_flight_data.csv (formatted flight data)
Output: cleaned_data.csv (formatted for model consumption)
        cleaned_data.csv.dvc (DVC metafile)
"""

# Load libraries
import pandas as pd

# Load the formatted flight data CSV file
df = pd.read_csv('formatted_flight_data.csv')

# Check for duplicate rows
duplicates = df.duplicated().sum()
if duplicates > 0:
    print(f"Found {duplicates} duplicate rows. Removing duplicates.")
    df = df.drop_duplicates()
else:
    print("No duplicate rows found.")

# Check for missing values
# print(df.isnull().sum())
# Fill missing float or integer values with 0
df.fillna(0, inplace=True)

# Check data types of each column
# print(df.info())
# Convert floats to integers where appropriate
float_columns = ['DEPARTURE_TIME', 'DEPARTURE_DELAY', 'ARRIVAL_TIME', 'ARRIVAL_DELAY']
df[float_columns] = df[float_columns].astype('int64')

# Filter for Atlanta (ATL) departures
# Print formatted data shape
# print(f"Formatted data shape:\n\tRows: {df.shape[0]}\n\tColumns: {df.shape[1]}")
df = df[df['ORG_AIRPORT'] == 'ATL']
# Print formatted data shape
# print(f"Filtered data shape:\n\tRows: {df.shape[0]}\n\tColumns: {df.shape[1]}")

# Save filtered data
df.to_csv('cleaned_data.csv', index=False)
```

D. MLflow Experiment

For `poly_regressor_Python_1.0.0.py`, I simply added the tracking of the artifacts, parameters, and metrics as per the instructions.

```
355
356 # TO DO: create an MLFlow run within the current experiment that logs the following as artifacts, parameters,
357 # or metrics, as appropriate, within the experiment:
358 # 1.  The informational log files generated from the import_data and clean_data scripts
359 # 2.  the input parameters (alpha and order) to the final regression against the test data
360 # 3.  the performance plot
361 # 4.  the model performance metrics (mean squared error and the average delay in minutes)
362
363 with mlflow.start_run(experiment_id = experiment.experiment_id, run_name = "Final Model - Test Data"):
364     # YOUR CODE GOES HERE
365     # 1.  The informational log files generated from the import_data and clean_data scripts
366     mlflow.log_artifact("polynomial_regression.txt")
367     mlflow.sklearn.log_model(ridgereg, "final_model")
368
369     # 2.  the input parameters (alpha and order) to the final regression against the test data
370     mlflow.log_param("alpha", parameters[0])
371     mlflow.log_param("order", parameters[1])
372
373     # 3.  the performance plot
374     mlflow.log_artifact("model_performance_test.jpg")
375
376     # 4.  the model performance metrics (mean squared error and the average delay in minutes)
377     mlflow.log_metric("mean_squared_error", score)
378     mlflow.log_metric("average_delay_minutes", np.sqrt(score))
379
380 mlflow.end_run()
381
382 logging.shutdown()
```

Below is a screenshot of `main.py`, which links `import_and_format.py`, `clean_and_filter.py`, and `poly_regressor_Python_1.0.0.py` for the `MLProject` file.

main.py 695 B

```
1  """
2  main.py - Run the complete flight delay prediction pipeline.
3  """
4
5  import subprocess
6  import sys
7  import os
8
9  def main():
10     """
11     Main function to run the entire pipeline.
12     """
13     try:
14         # Run the import and format script
15         subprocess.run([sys.executable, 'import_and_format.py'], check=True)
16
17         # Run the clean and filter script
18         subprocess.run([sys.executable, 'clean_and_filter.py'], check=True)
19
20         # Run the poly_regressor script
21         subprocess.run([sys.executable, 'poly_regressor_Python_1.0.0.py'], check=True)
22
23     except subprocess.CalledProcessError as e:
24         print(f"An error occurred while running the pipeline: {e}")
25         sys.exit(1)
```

E. MLProject Linking File

The `MLProject` file provides the project structure with entry points for the main pipeline and individual components.

See the screenshot below:

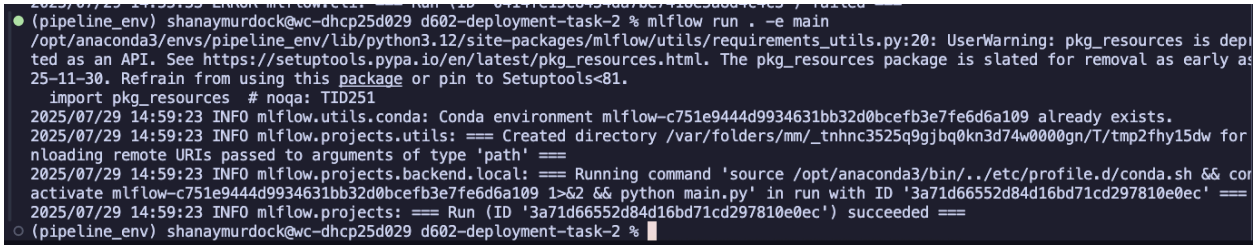
Add main.py and modify MLProject file
Shanay Murdock authored 6 hours ago

Code owners Assign users and groups as approvers for specific file changes. Learn more.

MLProject 410 B

```
name: flight_delay_prediction
conda_env: pipeline_env.yaml

entry_points:
    main:
        command: 'python main.py'

    import_data:
        command: 'python import_and_format_data.py'

    clean_data:
        command: 'python clean_and_filter.py'

    train_model:
        parameters:
            num_alphas: { type: int, default: 20 }
        command: 'python poly_regressor_Python_1.0.0.py {num_alphas}'

```

See the below screenshot to see the successful run of MLflow:



F. Explanation (Challenges and Solutions)

Challenge: Understanding DVC as a new concept and where to implement it. I had committed the file using Git and had to learn how to remove the tracking from Git.

Solution: Install and import DVC (Tran 2025, pp. 185-187). Remove the tracked file from Git, upload the original file to Git but commit the cleaned data file to DVC so that the cleaned data can be tracked during the model run.

Challenge: Formatting MLProject file

Solution: I had to understand it's a yaml file with no file extension. I formatted the file contents based on a lesson from *Introduction to MLflow* course on DataCamp (Bassler, n.d.).

Challenge: MLflow server already in use.

Solution: The instructions provided didn't cover what to do in the event that I had to walk away from the project and come back to it. I kept getting errors that the server was already in use but the MLflow UI wasn't working if I just went back to the site. I found an article with a command line script for stopping the server but the prompt didn't work. I had to restart the computer and reactivate environment.

G. Resources

Bassler, W. (n.d.) *Introduction to MLflow*. DataCamp. https://app.datacamp.com/learn/courses/introduction-to-mlflow

Tran, K. (2025) *Production Ready Data Science: From Prototyping to Production with Python*. Self published.

WGU Official Course Resources