Robert Pavlik

D602 – Deployment

QBN1 – Task 2: Data Production Pipeline

Introduction

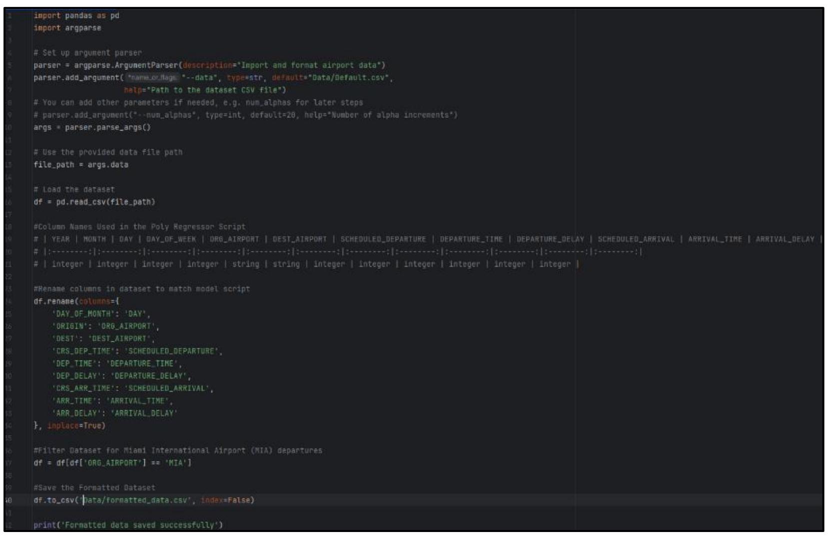
This project aimed to develop an end-to-end machine learning pipeline for predicting airport flight delays. I chose to do Miami, Florida Airport (MIA) using June 2024 data. The pipeline involved several steps: data importing, formatting, filtering, cleaning, and model training using a polynomial ridge regression model with MLflow for experiment tracking. The final code was integrated into a MLProject to enable reproducible runs and parameter tuning.

Requirements

A. Gitlab Repository – Task2_Branch

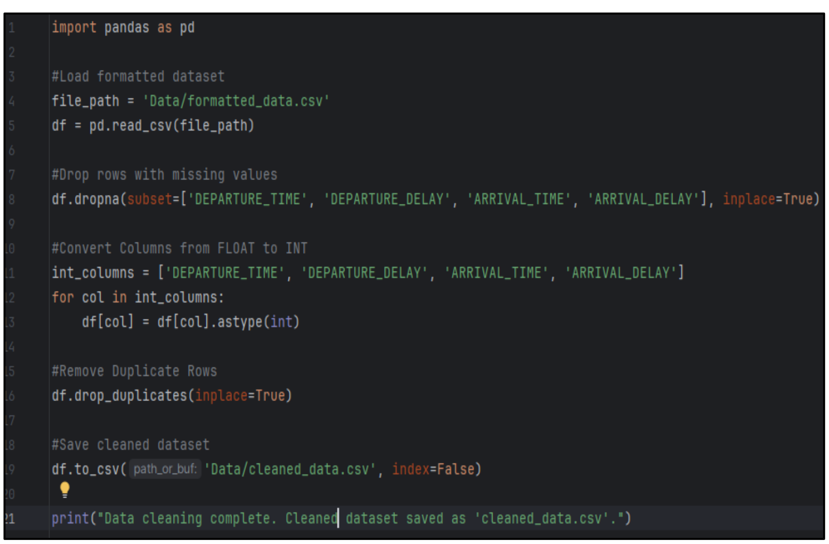
- https://gitlab.com/wgu-gitlab-environment/student-repos/rpavli5/d602-deployment-task-2.git

B. Import and Format Script



```python
import pandas as pd
import argparse

# Set up argument parser
parser = argparse.ArgumentParser(description="Import and format airport data")
parser.add_argument("--data", type=str, default="Data/Default.csv",
                    help="Path to the dataset CSV file")
# You can add other parameters if needed, e.g. num_alphas for later steps
# parser.add_argument("--num_alphas", type=int, default=20, help="Number of alpha increments")
args = parser.parse_args()

# Use the provided data file path
file_path = args.data

# Load the dataset
df = pd.read_csv(file_path)

#Column Names Used in the Poly Regressor Script
# | YEAR | MONTH | DAY | DAY_OF_WEEK | ORG_AIRPORT | DEST_AIRPORT | SCHEDULED_DEPARTURE | DEPARTURE_TIME | DEPARTURE_DELAY | SCHEDULED_ARRIVAL | ARRIVAL_TIME | ARRIVAL_DELAY |
# |:--------:|:--------:|:--------:|:--------:|:--------:|:--------:|:--------:|:--------:|:--------:|:--------:|:--------:|:--------:|
# | integer | integer | integer | integer | string | string | integer | integer | integer | integer | integer | integer |

#Rename columns in dataset to match model script
df.rename(columns={
    'DAY_OF_MONTH': 'DAY',
    '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'
}, inplace=True)

#Filter Dataset for Miami International Airport (MIA) departures
df = df[df['ORG_AIRPORT'] == 'MIA']

#Save the Formatted Dataset
df.to_csv('Data/formatted_data.csv', index=False)

print('Formatted data saved successfully')
```

C. Filter and Clean Script



```python
import pandas as pd

#Load formatted dataset
file_path = 'Data/formatted_data.csv'
df = pd.read_csv(file_path)

#Drop rows with missing values
df.dropna(subset=['DEPARTURE_TIME', 'DEPARTURE_DELAY', 'ARRIVAL_TIME', 'ARRIVAL_DELAY'], inplace=True)

#Convert Columns from FLOAT to INT
int_columns = ['DEPARTURE_TIME', 'DEPARTURE_DELAY', 'ARRIVAL_TIME', 'ARRIVAL_DELAY']
for col in int_columns:
    df[col] = df[col].astype(int)

#Remove Duplicate Rows
df.drop_duplicates(inplace=True)

#Save cleaned dataset
df.to_csv('Data/cleaned_data.csv', index=False)

print("Data cleaning complete. Cleaned dataset saved as 'cleaned_data.csv'.")
```

D. Poly_Regressor Script

```python
with mlflow.start_run(run_name = 'Test Data Run'):
    # Log input parameters
    mlflow.log_param( key: "alpha", parameters[0] / 10)
    mlflow.log_param( key: "polynomial_order", parameters[1])

    # Log model performance metrics
    mlflow.log_metric( key: "Test Data Mean Squared Error", score)
    mlflow.log_metric( key: "Test Data Average Delay", np.sqrt(score))

    # Log the performance plot artifact
    mlflow.log_artifact("Output/model_performance_test.jpg")

    # Log additional artifacts
    mlflow.log_artifact("Output/polynomial_regression.txt")
    mlflow.log_artifact("Data/cleaned_data.csv")

mlflow.end_run()

logging.shutdown()
```

E. MLProject File

```yaml
name: TestPipeline

conda_env: pipeline_env.yaml

entry_points:
    main:
        parameters:
            data: {type: str, default: cleaned_data.csv}
            num_alphas: {type: int, default: 20}
        command: "python Main.py {data} {num_alphas}"
    import_format:
        parameters:
            data: {type: str, default: cleaned_data.csv}
        command: "python Steps/StepB_Import_Format.py {data}"
    filter_clean:
        command: "python Steps/StepC_Filter_Clean.py"
    train_model:
        parameters:
            num_alphas: {type: int, default: 20}
        command: "python Steps/poly_regressor_Python_1.0.0.py {num_alphas}"
    run_all:
        parameters:
            data: {type: str, default: cleaned_data.csv}
            num_alphas: {type: int, default: 20}
        command: "python Main.py {data} {num_alphas}"
```

F. MLProject Pipeline Creation

The project was developed using Python and integrated with MLFlow to track model parameters, metrics, and artifacts. The overall goal was to create a reproducible pipeline that downloads, cleans, and filters data, and then runs a polynomial regression model to predict departure delays from a specified airport.

Data Import and Formatting:
I began by writing a Python script to import the raw flight data. This script reads data from a CSV file and reformats it to match the structure required by the regression model. Multiple iterations of the code were committed to GitLab, demonstrating a progression in functionality and robustness.

Data Filtering and Cleaning:
A separate script was developed to filter the dataset for departures from a chosen airport—in this

case, Miami International Airport (MIA). In addition to filtering by airport, I implemented further data cleaning steps, including:

- Dropping rows with missing or invalid delay values.
- Removing duplicate entries.
- Converting relevant columns from float to integer types where necessary.

Each stage was version-controlled and pushed to GitLab with incremental improvements.

MLFlow Experiment Integration:
The provided regression model (poly_regressor) was modified to integrate MLFlow for experiment tracking. In this script:

- Model parameters and performance metrics (mean squared error, average delay) were logged.
- Artifacts, such as the training log files and a performance plot, were saved.
- The MLFlow experiment was made reproducible by linking the run context to a YAML-based MLProject file.

Several iterations of this script were committed to GitLab, showing iterative improvements and troubleshooting.

MLProject File Creation:
To connect the data import/cleaning scripts with the MLFlow experiment, I created an MLProject YAML file that:

- Specifies the entry points for both the data processing script and the MLFlow experiment script.
- Defines parameters (including their types and default values) that can be passed during runtime.
- Enables seamless execution of the entire pipeline from the command line using mlflow run.

The pipeline is designed so that other analysts can easily extend the work to different airports by modifying the parameters and re-running the pipeline. The MLProject file and related scripts are version-controlled in the GitLab repository, ensuring full traceability of changes.

Challenges Encountered and Solutions:

- **MLFlow Run Context:**
  Managing the MLFlow run context when running via mlflow run was challenging. Initially, I attempted to manually retrieve or start an active run within the script, which led to errors such as “Run not found.” This was resolved by removing explicit run management and relying on the MLFlow CLI to handle the run context automatically.
- **Parameter Passing:**
  Integrating the scripts to accept parameters via the command line initially resulted in Key Error: 'data' errors. I addressed this by updating the MLProject file to include a parameter

section for the entry points and ensuring that the Python scripts correctly accepted these parameters, specifying types and default values.

Conclusion:

By addressing these challenges through careful updates to the code, MLProject file, and environment management, the final MLProject pipeline was successfully executed using MLFlow. The pipeline now supports dynamic parameter substitution, reproducible Conda environments, and detailed experiment tracking via the MLFlow UI. A screenshot of the successful MLProject run is included below.