Airflow for Athlete Wearable Teach- Project 3

Reback Operations- Datawarehouse team

Student: Maheen Gul Khattak (224031218)

Introduction and Project Overview

Project Purpose and Objectives:

The purpose and objective of this project is to establish a data pipeline with Apache Airflow to streamline data processing activities for the Data Warehouse Team. The objective is to create a centralized data management system that not only serves the purpose of processing and cleaning the data but also allows teams to upload and store datasets for the purpose of additional processing and combination.

Using Airflow, teams can optimize data management operations, guaranteeing consistency, precision and availability of data. The main **objectives** consist of:

- Establishing an automated data pipeline with Airflow for overseeing data processing activities.
- Carrying out data cleaning procedures to address absent values and formatting discrepancies.
- Compiling the data to deliver monthly overviews, encompassing total runs, average laps, maximum laps and minimum laps.
- Establishing a data storage system that allows other teams or departments to upload their datasets, which care subsequently processed and arranged for future use.
- Creating a data repository to store cleaned and aggregated data in CSV format, functioning as a centralized warehouse that can be accessed by other teams for their analysis.

This project illustrates how Airflow serves as a data orchestration tool allowing various users to upload datasets that are subsequently processed, cleaned and organized systematically for easy retrieval and future analysis.

Data Source Overview (Garmin Run Data):

The dataset utilized in this project consists of Garmin running information gathered from wearable gadgets. This information is organized in CSV format and contains important running metrics like activity type, data, distance, duration, heart rate and lap details.

This information is processed to address absent values, standardize data columns into a uniform format and set it up for additional aggregation. After processing, the organized and combined data acts as a fundamental dataset for upcoming analysis and reporting.

Tools and Technologies

Tools/Technology	Purpose	How it was used
Apache Airflow	Orchestration of data pipelines.	Created DAGs in Python to streamline data processing activities, such as reading, cleaning, summarizing and aggregating Garmin running data.
Docker	Containerization.	Implemented Airflow services (webserver, scheduler, worker, triggerer) in containers by utilizing Docker compose. Established a uniform setting for executing Airflow tasks.
Windows Command Prompt/ Windows PowerShell	Execution of command and the purpose of service management.	Employed to traverse project directories, run Docker Compose commands and oversee container status.
Python	For the purpose of data processing and task development.	Utilized Python to create DAGs. Data processing activities were specified utilizing libraries like pandas for data handling and logging for overseeing execution stages.
Pandas Library	To manipulate and analyse data.	Handled the Garmin dataset for data cleaning, creating summary statistics and conducting data aggregation.
Logging Library	To monitor and debug tasks.	Documented task execution information, encompassing data structure, data modifications and output creation.

Environmental setup and Configuration

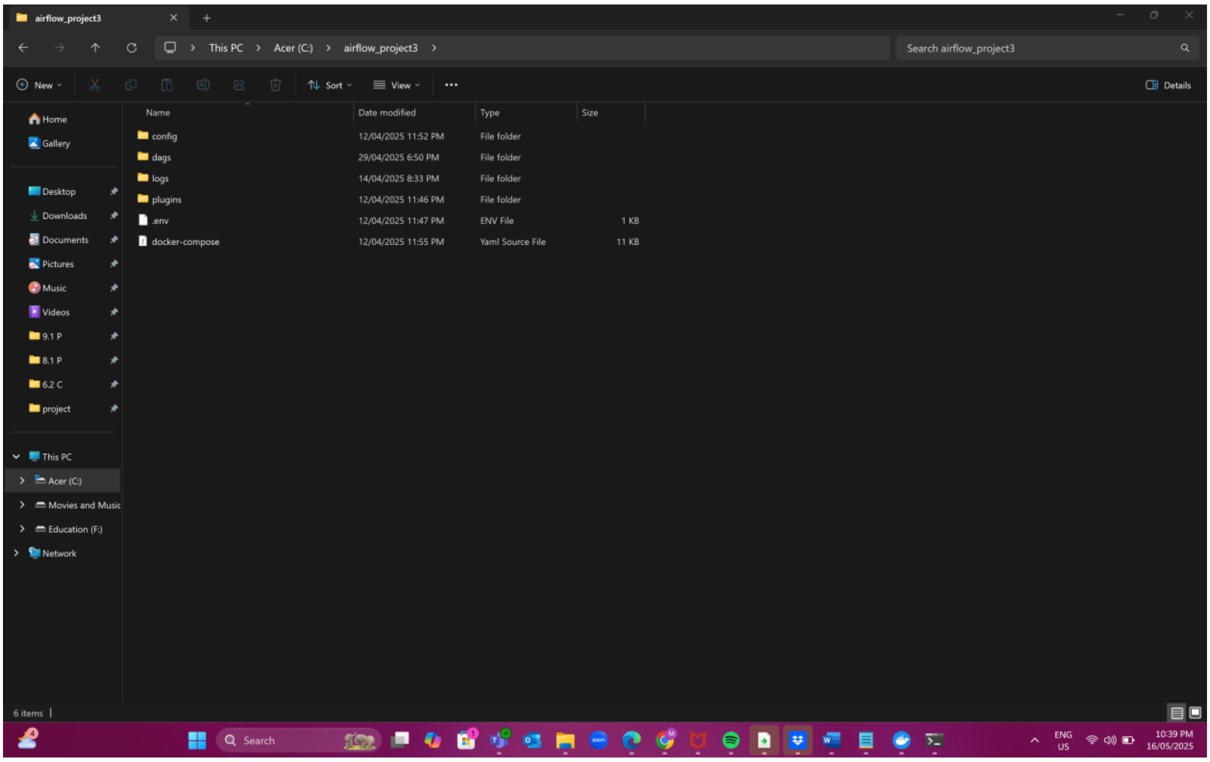
Step 1: Create the Project Directory:

- After installing docker and getting it set up (please refer to https://www.youtube.com/watch?v=BuGEGM_elXY), the next step is to
- Establish the Airflow environment involved creating a specific project directory with these commands:

  Mkdir C:\airflow_project3
  Cd C:\airflow_project3

  Note: remember to run Window Powershell as administrator for this command

This directory functioned as the work area for the Airflow project and contained all essential files, such as the dags folder, docker-compose.yml and data files.



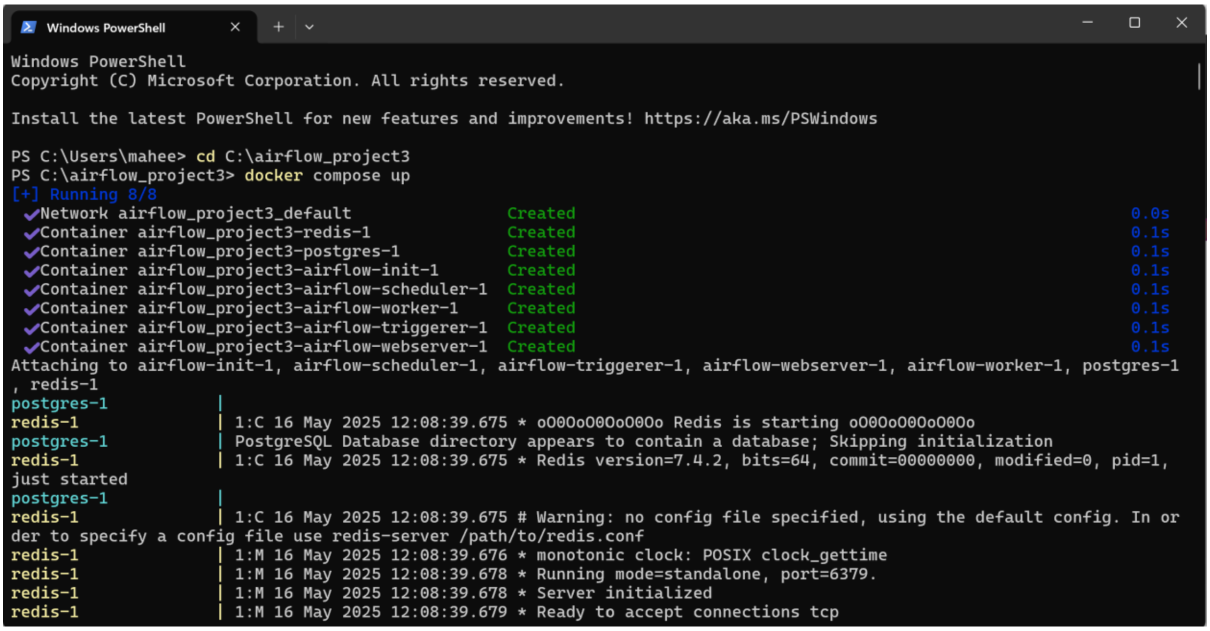
Step 2: Setting up Docker and Docker compose

We used Docker Compose to deploy Airflow with Docker (make sure your docker is running before trying to get your AIR FLOW to work through the commands below). The command run to initiate the container was:

C:\airflow_project3

docker compose up

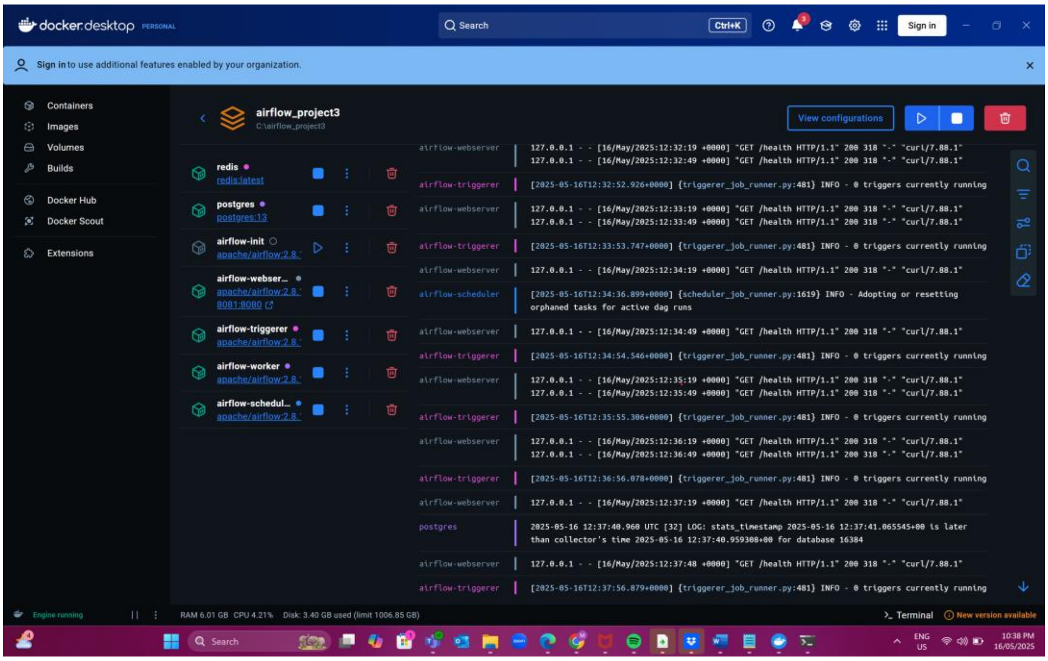
Note: remember to run Window Powershell as administrator for this command



This command launched several containers, such as webserver, scheduler, worker. Triggerer, postgres and redis. The console output confirms the successful creation and execution of these containers.

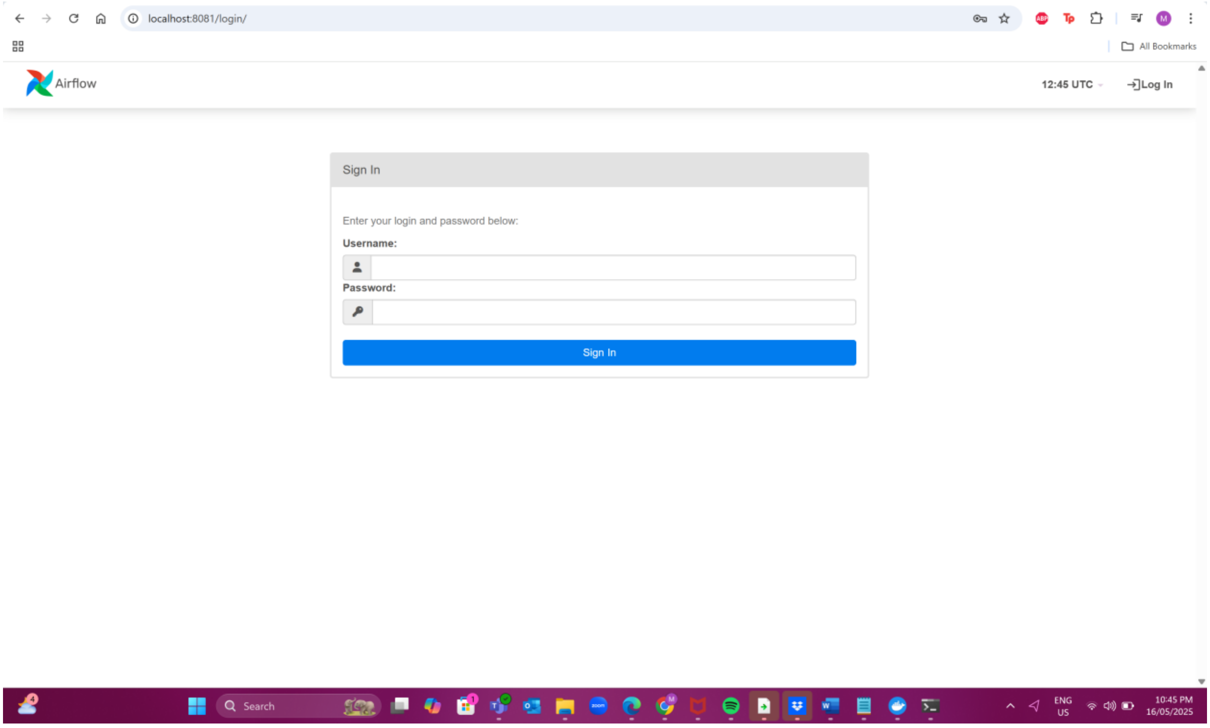
Step 3: Monitoring Containers in Docker Desktop

- Docker Desktop offered a graphical representation of all active containers.
- The airflow_project3 container group showed every Airflow service along with its status, encompassing memory and CPU utilization.
- This graphical depiction assisted in confirming that all elements were functioning as anticipated.
- Docker Desktop offered a graphical representation of all active containers.



Step 4: Airflow webserver access:

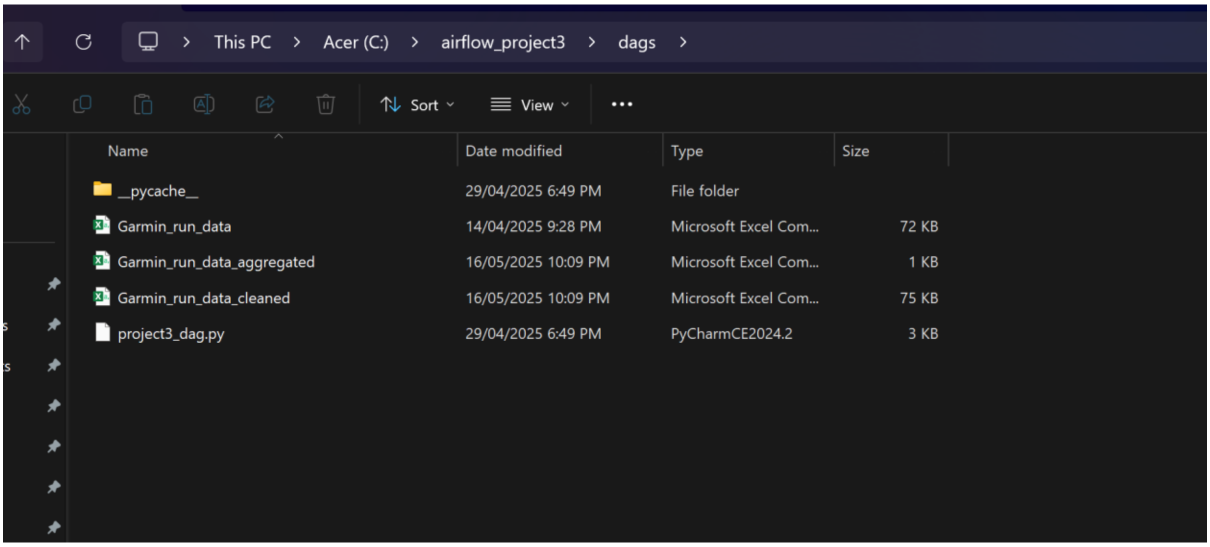
- The airflow_project3 container group showed every Airflow service along with its status, encompassing memory and CPU utilization.
- Once everything is running, AirFlow will be running on the assigned port number and will ask the user to give a username and password as shown below.



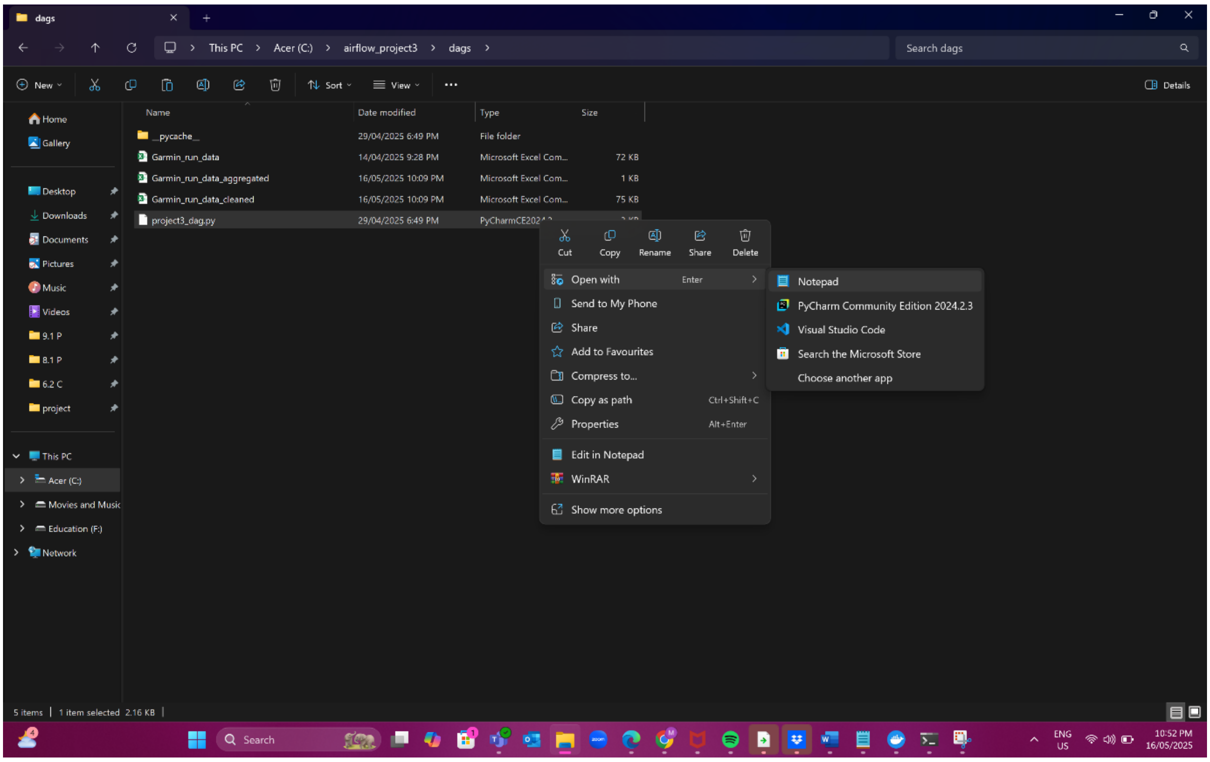
Implementation and development Process

- In this part, I will analyse the gradual creation of the DAG, outlining the libraires utilized, the function of each task, and the way tasks are linked. The DAG was developed in Python, employing libraries like pandas for data handling, logging for oversight and Airflow's PythonOperator for executing tasks.

- The dags folder is the most important in the airflow_project3 folder, this is where a user may use the file project3_dag.py to write the code and control the tasks in the airflow in order to tell AirFlow what it can do with the data.



- In order to write code in the project2_dag.py a user will have to



- Right click project3_dap.py
- Go to open with.
- Click on notebook.
- The user may then write the code as they desire, depending on how the user wants the tasks to be scheduled, in my case I have written the code as shown below.

Writing the DAG Using Python:

The DAG is defined using Python and consist of three primary tasks which include:

- **read_and_clean_data:** Reads and cleans the datasets.
- **summary_statistics:** Produces summary statistics for the processed data.
- **aggregate_data:** Combines data monthly and computes essential metrics.

File Edit View

```
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime
import pandas as pd
import logging

# Task function 1: Read and clean the Garmin run data
def read_and_clean_data():
    df = pd.read_csv("/opt/airflow/dags/Garmin_run_data.csv")
    logging.info(f"Original shape: {df.shape}")

    df['Date'] = pd.to_datetime(df['Date'], errors='coerce')
    logging.info(f"Cleaned shape: {df.shape}")
    logging.info("First 5 rows:\n%s", df.head())

    df.to_csv("/opt/airflow/dags/Garmin_run_data_cleaned.csv", index=False)
    logging.info("Saved cleaned data to Garmin_run_data_cleaned.csv")

# Task function 2: Generate summary statistics
def summary_statistics():
    df = pd.read_csv("/opt/airflow/dags/Garmin_run_data_cleaned.csv")
    summary = df.describe(include='all')
    logging.info("Summary statistics:\n%s", summary)
```

Ln 63, Col 34 | 2,146 characters | 100% | Windows (CRLF) | UTF-8

Explanation:

- This section specifies the function read_and_clean_data.
- It imports the original dataset, transforms the “Date” coloumn to datatime formate, and saves the processed data.
- Logging is utilized to document both raw and redefined state of the data for oversight reasons.

File Edit View

```
# Task function 3: Aggregate data by month
def aggregate_data():
    df = pd.read_csv("/opt/airflow/dags/Garmin_run_data_cleaned.csv")
    df['Date'] = pd.to_datetime(df['Date'], errors='coerce')
    df['Month'] = df['Date'].dt.to_period('M')

    aggregated = df.groupby('Month').agg(
        total_runs=('Activity Type', 'count'),
        average_laps=('Number of Laps', 'mean'),
        max_laps=('Number of Laps', 'max'),
        min_laps=('Number of Laps', 'min')
    ).reset_index()

    logging.info("Aggregated Monthly Data:\n%s", aggregated)

    aggregated.to_csv("/opt/airflow/dags/Garmin_run_data_aggregated.csv",
index=False)
    logging.info("Saved monthly aggregated data to
Garmin_run_data_aggregated.csv")

# Define the DAG
with DAG(
    dag_id="project3_dag",
    start_date=datetime(2025, 4, 1),
```

Ln 19, Col 23 | 2,146 characters | 100% | Windows (CRLF) | UTF-8

Explanation:

- This section outlines the summary_statistics function.
- It processes the cleaned dataset and produces summary statistics, such as count, mean, standard deviation and minimum/maximum values for every column.

File Edit View

```
    dag_id="project3_dag",
    start_date=datetime(2025, 4, 1),
    schedule_interval="@daily",
    catchup=False,
    tags=["project3"]
) as dag:

    task1 = PythonOperator(
        task_id="read_and_clean_data",
        python_callable=read_and_clean_data
    )

    task2 = PythonOperator(
        task_id="summary_statistics",
        python_callable=summary_statistics
    )

    task3 = PythonOperator(
        task_id="aggregate_data",
        python_callable=aggregate_data
    )

    task1 >> task2 >> task3
```

Ln 41, Col 75 | 2,146 characters | 100% | Windows (CRLF) | UTF-8

Explanation:

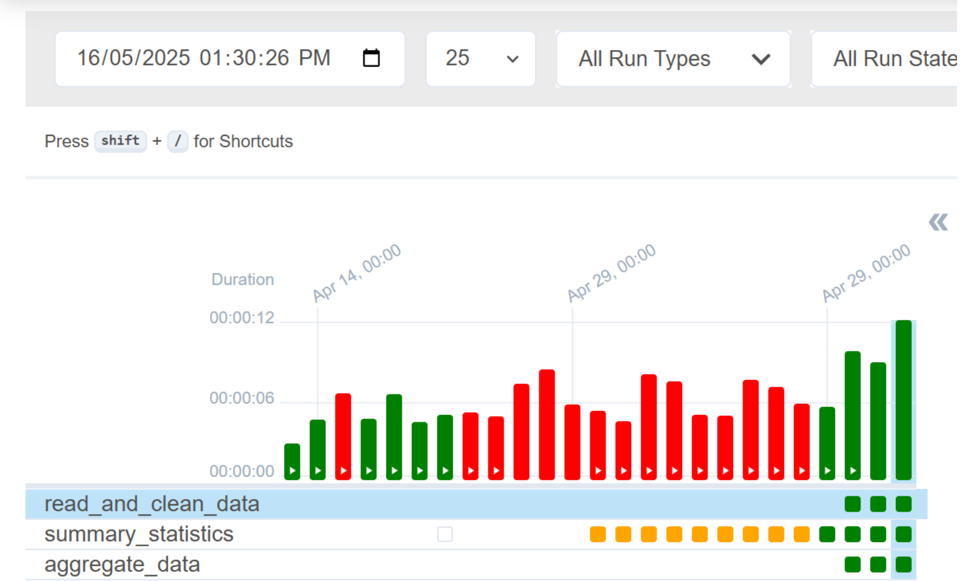
- This section outlines the aggregate_data_function.
- It computes essential metrics for every month, including total runs, average laps, maximum laps and minimum laps.
- The compiled data I stored in a new CSV file for additional analysis.

Overall summary of Code:

- The DAG is defined with the ID project3_dag.
- The schedule interval is set to @daily, meaning the DAG will run everyday.
- The three tasks are arranged in a sequential order using the >> operator: read_and_clean_data → summary statistics → aggregate_data.

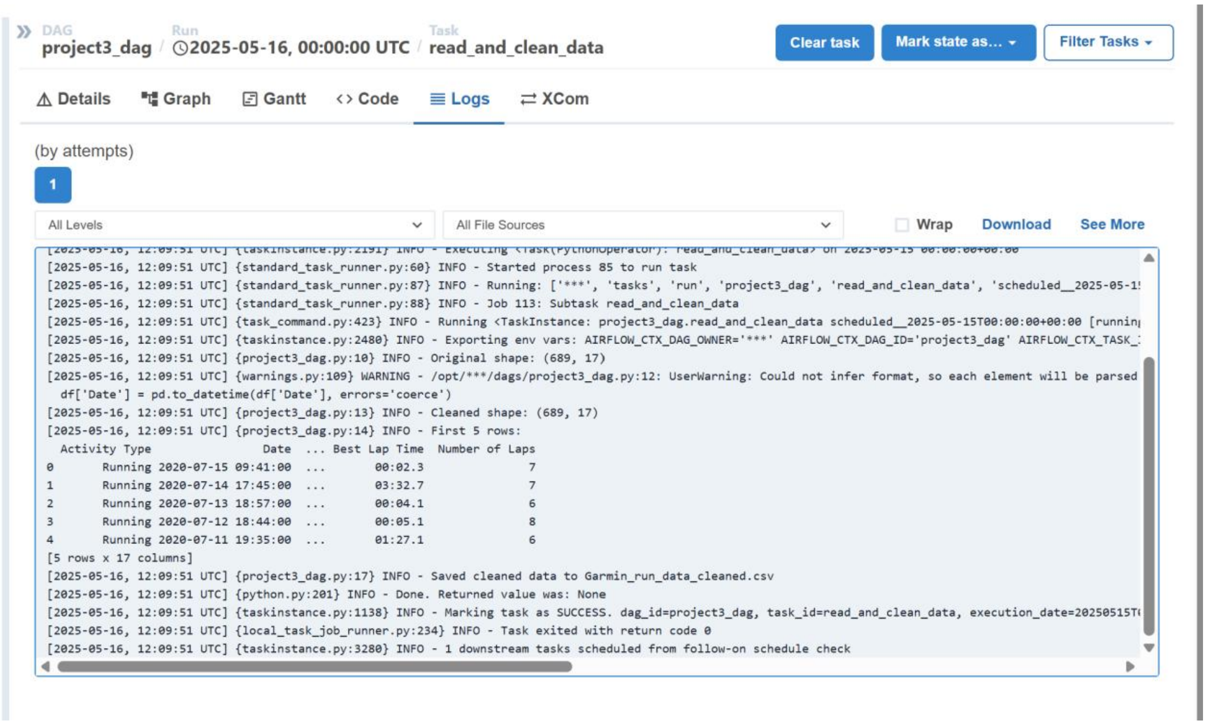
Data Processing and Aggregation

- In the screenshot below it shows the total number of times the tasks were run and how many times did the tasks fail to run or succeeded.



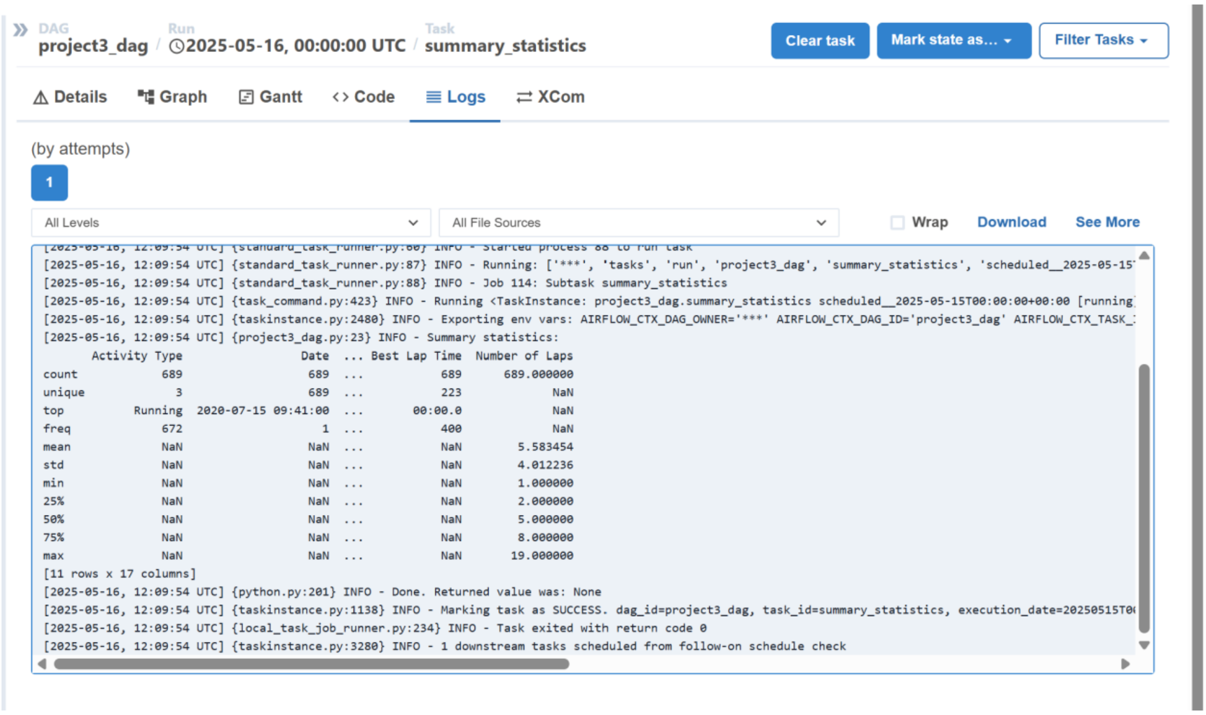
1. **Reading and cleaning the data.**

- **Objective:** The objective of this task is to analyze the raw Garmin run data and process it by transforming the “Date” column into a datetime format. This phase was crucial to maintain uniformity in data formats to facilitate later aggregation by month.
- **Implementation:**
- The raw data was read through the pandas.read_csv() function.
- The “Date” coloumn was transformed into datetime format usind pd.to_datettime() with error coercion to manage any irregular data formats.
- A log entry was created to show the initial data structure and the first 5 wows of the refined data.



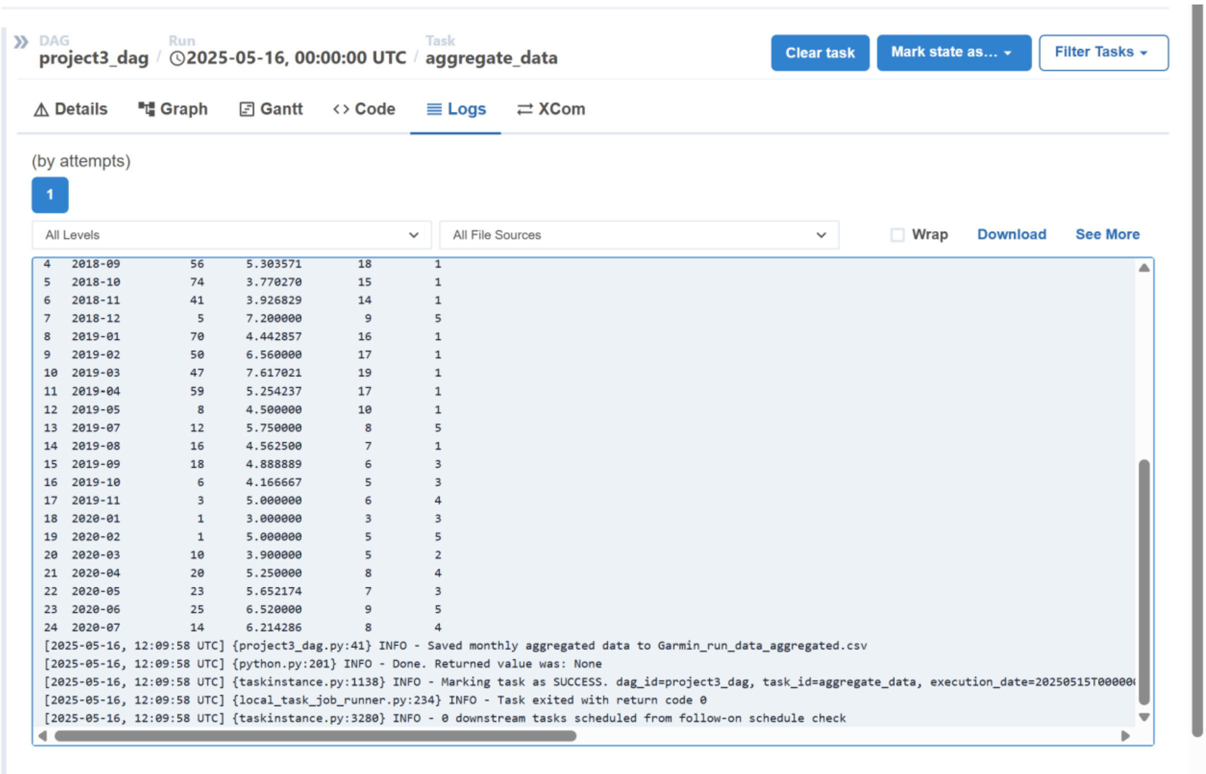
2. **Summary Statistics:**

- **Objective:** The objective of this task was to produce descriptive statistics for the purified Garmin data in order to offer insights into the dataset, including counts, unique values and fundamental statistics.
- **Implementation:**
- The cleaned data was analyzed once more, and describe() was utilized to produce summary statistics, which encompassed counts, means, standard deviations and quartile values.
- The summary statistics were recorded for additional analysis.



3. **Aggregating Data by Month:**

- **Objective:** The aim of this task was to compile the data monthly to offer a unified perspective on the number of runs, average laps and the minimum/maximum laps for every month.
- **Implementation:**
- The cleaned data was analyzed to retrieve the “Month” from the “Date” coloumn by means of .dt.to_period(‘M’).
- The data was subsequently combined with groupby() to determine the total runs, average laps, maximum laps and minimum laps for every month.
- The outcomes were recorded and stored as a new CSV file for additional analysis.



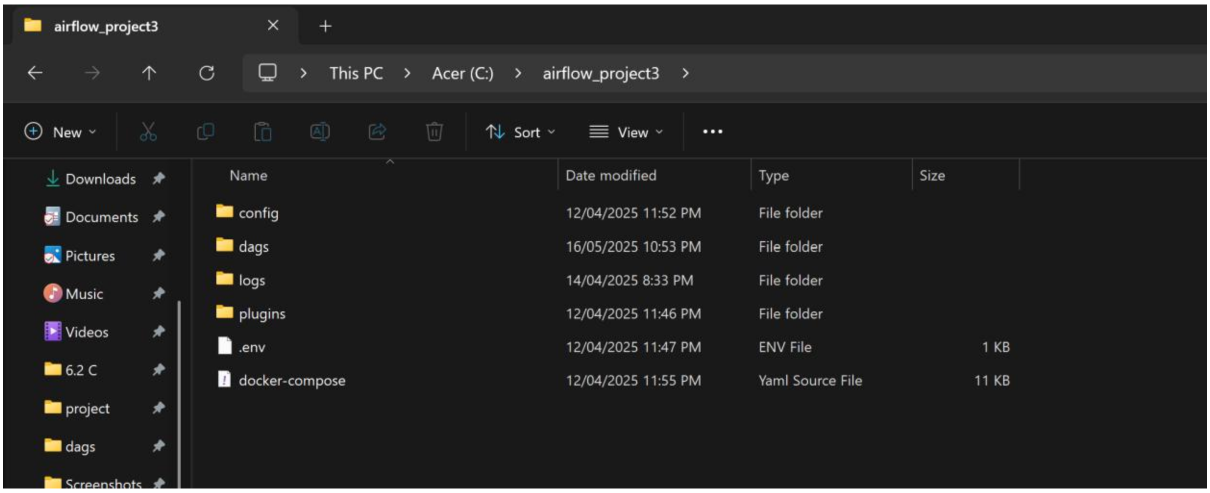
Data storage and Access

In this section, I will describe how the organized and compiled data is kept and how it can be reached by other teams. The processed information is stored as CSV files in a specific folder, which is included in the Airflow project layout. Here are the specifics of the data storage and the method to access it.

1. **Airflow Project Directory Structure - C:\airflow_project3**

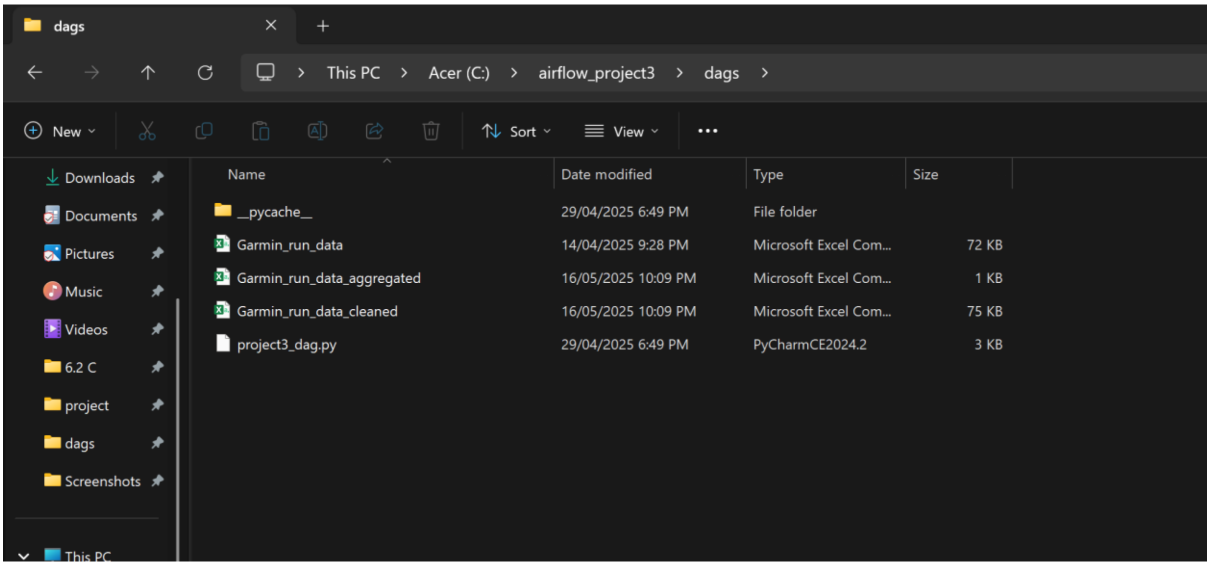
- The screenshot below shows the layout of the Airflow project directory, emphasizing the crucial folders and files needed for establishing and running the Airflow environment.
- **Dags:** Saves the Python script and output data files.
- **Logs:** Includes logs produced while running DAG tasks, beneficial for troubleshooting and oversight.
- **Plugins:** Designated for personalized plugins and operators.
- **Docker-compose.yml:** Configuration file utilized to launch Airflow and its associated services (webserver, scheduler, worker, triggerer, Redis and PostgreSQL) via Docker Compose.
- **.env:** Variables from the environment needed for setting up Airflow components.

Explanation: This setup guarantees that all elements are arranged and easily manageable. The DAGs directory is especially important for retrieving the DAG script.



2. **Data Files Directory - C:\airflow_project3\dags**
   Purpose: This screenshot displays the folder in which the processed and aggregated data files are stored. The primary files shown are:
   - **Garmin_run_data_cleaned.csv:** This document holds the cleaned Garmin running data following data transformation and elimination of discrepancies.
   - **Garmin_run_data_aggregated.csv:** This document includes the monthly statistics, such as total runs, average laps, maximum laps and minimum laps.
   - **Garmin_run_data.csv:** This is the initial dataset prior to any cleaning or aggregation.
   - **Project3_dag.py:** The Python script that includes the definition of the Airflow DAG and its tasks.

Explanation: These files are kept in the dags directory inside the Airflow project folder. The data files that are cleaned and aggregated are automatically created and stored after executing the Airflow DAG. Keeping these files in a central location allowed other teams to conveniently access them for additional analysis or reporting.



Challenges and Solutions

While creating the data processing pipeline with Apache Airflow, multiple challenges were faces. Here is a recap of these difficulties and the measure taken.

1. **Setting Up Docker and Airflow:**
    - **Challenge:** Configuring the Docker containers and confirming that all Airflow services (webserver, scheduler, triggerer, worker) were correctly launched. There were few problems with certain containers failing to start or staying in pending status.
    - **Solution:** The problem was fixed by carefully reviewing the docker-compose.yaml configuration file for syntax mistakes and confirming that all environment variables were properly set in the .env file. Furthermore, executing the command docker compose up in the C:\airflow_project3 folder successfully set up all containers.
2. **File path Issues in Airflow Containers:**
    - **Challenge:** In the process of reading and cleaning the data, the DAG was unable to find the data file (Garmin_run_data.csv). This occurred because of erroneous file paths in the container environment.
    - **Solution:** The accurate file path/ opt/airflow/dags/ was verified, and the data file was relocated to the correct directory. Furthermore, the route was specifically mentioned in the DAG code to avoid additional path-related problems. The log output demonstrates the successful completion of the read and clean data task.
3. **Debugging DAG Errors:**
    - **Challenge:** Several task failures happened during the early executions of the DAG. Especially with the aggregate_data task. This was primarily because of absent or incorrectly formatted data entries in the CSV file that led to failures in pandas operation.
    - **Solution:** The issues were fixed by incorporating error handling in the aggregate_data function with errors= "coerce" while converting the data column into a datetime format. This enabled invalid dates to be managed without interrupting the task progression.
4. **Airflow Webserver Access:**
    - **Challenge:** At times, the Airflow webserver was unreachable even though all containers were operational. This was probably caused by incompatible ports or delays in starting the containers.
    - **Solution:** The problem was resolved by restarting the containers with docker compose down and then docker compose up. Observing the console output verified the successful initiation of the webserver at localhost:8081.
5. **Resource Utilization and Container Overload:**
    - **Challenge:** While running several DAG executions, there was a notable spike in container CPU usage, leading to slow performance and intermittent crashes.
    - **Solution:** Resource distribution was enhanced by modifying container resource limits in the docker-compose.yaml file. Unneeded containers were halted to free up memory and processing capabilities.

Conclusion and Future Implications:

My main goal was to create an organized data processing pipeline that allows other teams to efficiently handle and arrange their data (particularly project 3). The existing implementation effectively showcased how air flow can be utilized to automate data cleaning aggregation statistics, establishing a fundamental framework for data processing. The data pipeline guarantees data consistency and offers a systematic method for additional data processing and analysis.

Future Implications and Next steps:

1. **Implementing a data upload system for Project 3 team members:** The next significant phase of the project is to create a strong data upload system that enables Project three members to effortlessly integrate new databases into their Airflow pipeline. This might include developing a web interface or a centralized network drive for team members to upload new data files. This Airflow DAG can subsequently be set up to automatically identify and handle these new files guaranteeing that the data warehouse stays up-to-date.

2. **Centralized data access and management:** Although the existing setup keeps processed data on local CSV files and more scalable solution would involve connecting to a centralized database. This would allow multiple teams to access the cleaned and aggregated data without having to navigate through the Airflow directory structure. Additionally, by setting up the user permissions and access control would enhance data security and integrity.

3. **Error handling and task monitoring** is in place, enhancing error handling would increase task reliability. This involves recording particular exceptions (e.g. missing data files, data forma mismatches) and notifying team through email or messaging services whenever a task fails.

4. **Scalability and automation:** With the increase in data volume, it will be crucial to optimize the pipeline for scalability. This may require executing parallel processing duties, planning data backups and integrating cloud storage options for prolonged data preservation.

5. **Data archiving and retention:**
   Creating a data archiving system to save historical data in a distinct directory or cloud storage would keep the project directory organized and free from clutter. Automated processes can be set up to transfer outdates data files to a specified archive location, maintaining a tidy orderly workspace.

Reference list

ProgrammingKnowledge (2025). *How to Install Docker on Windows & Linux | Step-by-Step Guide for Beginners*. [online] YouTube. Available at: https://www.youtube.com/watch?v=BuGEGM_elXY [Accessed 16 May 2025].