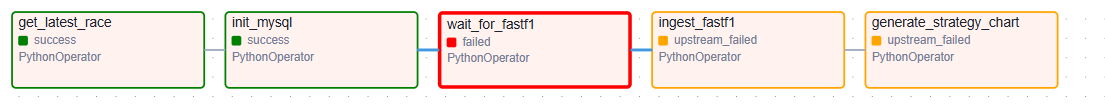
**Formula 1 Data Engineering Pipeline Documentation**

**Introduction**

This project demonstrates the design and implementation of an end-to-end data engineering pipeline using Formula 1 telemetry and timing data. The goal was to replicate a real-world ETL process:

* Extracting raw race data from the FastF1 API
* Transforming and cleaning the data into structured tables
* Loading the processed data into a MySQL database
* Orchestrating the workflow with Apache Airflow running in a Dockerized environment



* A chart of a formula one race

  AI-generated content may be incorrect.Visualizing insights such as tyre degradation, stint strategies, and pit stops through Python (Matplotlib/Seaborn) dashboards, with later expansion to Grafana/Power BI

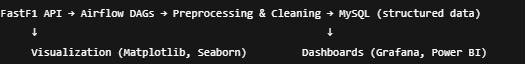
The pipeline showcases skills in ETL, CI/CD principles, orchestration, data preprocessing, data modelling, and visualization, all of which are core responsibilities of a modern Data Engineer.

**System Architecture**

The pipeline follows a containerized microservices architecture, defined in a docker-compose.yaml file.

* Airflow Scheduler & Webserver – orchestrates the ingestion, transformation, and visualization tasks
* MySQL Database – stores cleaned race data in normalized relational tables  
  A screenshot of a computer

  AI-generated content may be incorrect.
* Grafana / Power BI (future) – front-end dashboards for business users
* Python ETL scripts – handle extraction from FastF1, preprocessing, and data quality checks
* Matplotlib/Seaborn Dashboards – current implementation of strategy and degradation visualization charts

Workflow Diagram:  


**ETL Pipeline Design**

1. Extraction

* Data is sourced from the FastF1 API, which provides race timing, driver, and tyre compound information.
* Airflow tasks trigger ingestion automatically, either on schedule (after a race weekend) or on-demand.
* A caching layer (fastf1\_cache) ensures efficient repeated queries.

1. Transformation & Preprocessing

Raw FastF1 data contains pit laps, invalid laps, and inconsistent timing fields.

* Lap times are standardized into milliseconds for easier calculations.
* Missing values (NaN) are handled via null checks.
* Outlier laps (e.g., pit stop laps, laps under safety car) are excluded from degradation analysis.
* Data is normalized into relational tables:
  + sessions → metadata (track, year, type, weather, safety cars)
  + drivers → driver details
  + laps → per-lap timing data linked by session and driver IDs

This setup makes sure the database is well-organized, with clear links between sessions, drivers, and laps so the data stays accurate and consistent.

1. Loading

* Cleaned data is bulk inserted into MySQL using SQLAlchemy.
* Foreign keys enforce relational integrity.
* ON DUPLICATE KEY UPDATE ensures idempotent runs - the DAG can re-ingest without duplicating rows.

**Workflow Orchestration with Airflow**

Apache Airflow is used to implement ETL orchestration and CI/CD best practices.

1. DAG (Directed Acyclic Graph) Structure

The DAG fastf1\_ingest\_mysql consists of the following tasks:

1. Init MySQL – drops/recreates schema
2. Wait for FastF1 – checks if data for the latest race is published
3. Ingest FastF1 – extracts, cleans, and loads data into MySQL
4. Generate Strategy Chart – produces stint visualization plots

2. Scheduling & Sensors

* The DAG runs daily and uses a FastF1 sensor task to detect when new race data is published.
* This prevents failed runs and demonstrates event-driven data pipelines.

**3. CI/CD Concepts**

* **Versioned with Git + Docker** → Same setup can run anywhere, changes are tracked.  
  *This makes the pipeline easy to share, test, and maintain.*
* **Safe to rerun** → Ingestion avoids duplicates or broken data.  
  *If something fails, you can rerun without damaging the database.*
* **Containerized setup** → Airflow, MySQL, Grafana run like a real production system.  
  *Each service is isolated but works together smoothly.*

**Data Visualization & Analytics**

1. Strategy Charts

* Generated with Matplotlib and stored in an Airflow-mounted output directory.
* Visualizes stint length, tyre compound choice, and pit stop laps per driver.
* Mimics professional F1 team race strategy dashboards.

A graph of a race

AI-generated content may be incorrect.

2. Tyre Degradation Analysis

* Degradation curves are plotted for each driver and across the field.
* Average lap times by compound highlight performance trade-offs between Soft, Medium, and Hard tyres.
* Outlier removal ensures trends reflect competitive laps, not pit in/out noise.
* Shaded error bands (± standard deviation) illustrate variability, echoing Pirelli’s official graphics.

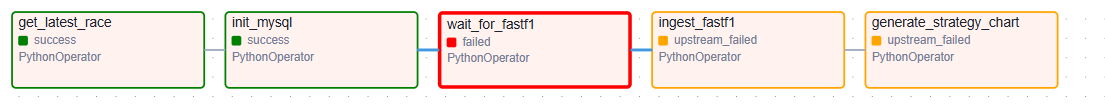
A graph showing the growth of the oil prices

AI-generated content may be incorrect.A graph of a number of lines

AI-generated content may be incorrect.

Data Quality & Preprocessing Techniques

* Schema validation – Primary and foreign keys in MySQL ensure sessions, drivers, and laps link correctly. This prevents broken or inconsistent records from being stored.  
  A diagram of a schema

  AI-generated content may be incorrect.
* Null handling – Missing values (like incomplete lap times) are either dropped if unusable, or replaced with safe defaults, so analyses don’t break.
* Outlier detection – Abnormally slow laps (e.g., pit stops or in/out laps) are detected using statistical thresholds and excluded from averages. This keeps performance metrics realistic.
* Standardization – All lap times are stored in milliseconds. Having a single format simplifies comparisons, calculations, and visualizations across different datasets.
* Logging & monitoring – Airflow automatically logs each run of the pipeline, showing whether ingestion and preprocessing steps succeeded or failed, making issues easy to spot and fix.  
  

**Skills Demonstrated**

* ETL & Data Pipelines – Designed and implemented an end-to-end data pipeline: extracting raw telemetry and timing data from the FastF1 API, transforming it into clean tabular structures, and loading it into a relational database for downstream analytics.
* Workflow Orchestration – Built Airflow DAGs that automate ingestion and transformation, with scheduling, retries for failed tasks, and modular task dependencies that reflect real-world production workflows.
* Database Engineering – Created a relational model linking sessions, drivers, and laps; enforced integrity constraints to prevent bad data; and optimized queries to support analytics use cases like lap time comparisons and tyre degradation analysis.
* Data Preprocessing & Cleaning – Applied techniques such as handling null values, detecting and removing outliers (e.g., pit stop laps), and standardizing time formats (ms → mm:ss.sss) to ensure clean, accurate analytics.
* Visualization & Analytics – Produced exploratory visualizations using Matplotlib and Seaborn (e.g., tyre degradation curves, lap time distributions) and built interactive dashboards in Grafana/Power BI to make results accessible to decision-makers.
* CI/CD Practices – Used Git for version control and Docker for containerization, ensuring reproducibility across environments. Implemented idempotent ingestion so that pipelines can be rerun safely without duplicating or corrupting data.
* Scalability – Designed Airflow tasks to be modular and extendable, allowing the pipeline to scale to multiple races, entire seasons, or even integrate additional APIs and cloud-based databases with minimal rework.

**Conclusion**

This project delivered a working Formula 1 analytics pipeline that simulates how real-world data engineering systems operate.

What was done & achieved:

* End-to-end pipeline built: Data was ingested directly from the FastF1 API, transformed, structured, and stored into a MySQL database.
* Relational modelling: The database was organized into sessions, drivers, and laps tables, ensuring clean links between race data.
* Automation with Airflow: Ingestion and preprocessing were orchestrated using Airflow DAGs, making the pipeline repeatable and reliable.
* Containerized deployment: Using Docker ensured the same setup could run anywhere, mirroring production-style workflows.
* Data quality enforced: Techniques such as schema validation, null handling, and outlier detection kept the data consistent and trustworthy.
* Visualization: Grafana dashboards were connected to MySQL, and Power BI exploration showed meaningful insights like lap times, tyre strategies, and driver comparisons.

Future extensions:

* Richer dashboards: Deeper visualization in Grafana or Power BI (e.g., pit stop strategies, tyre degradation comparisons, driver performance trends).
* Continuous integration/deployment (CI/CD): Automating pipeline tests and deployments using GitHub Actions.
* Monitoring & alerting: Adding checks for failed DAG runs, data anomalies, or API issues, with notifications.
* Scalability: Expanding to cover multiple races, seasons, and streaming data sources for near real-time analysis.