# Traffic-Accidents-ETL Project

**Performed by:**

- Isabella Pérez C.  - 2230603
- Luz Angela Carabali M. -  2230652
- Nicolas Peña I. -  2232049

**Version:** 3.0

**Update Date:** May 2025

**Introduction:**

The Traffic-Accidents-ETL project aims to design and implement an Extraction, Transformation, and Loading (ETL) pipeline to process a dataset of 209,306 traffic accident records collected between 2018 and 2025.

**Source:**

The dataset consists of 209,306 records related to traffic accidents from https://www.kaggle.com/datasets/oktayrdeki/traffic-accidents.

## Technologies Used

- **Language and Processing:** Python 3.12, Pandas, NumPy.
- **Visualization:** Matplotlib, Seaborn, Power BI Desktop, Streamlit.
- **Database:** PostgreSQL 15, psycopg2-binary, SQLAlchemy.
- **Orchestration:** Apache Airflow.
- **Streaming:** Apache Kafka.
- **Containerization:** Docker, Docker Compose.
- **Notebooks:** Jupyter Notebook.
- **Version Control:** Git, GitHub.
- **Testing:** pytest.
- **Data Validation:** Great Expectations.

# 1. Connection Between Deliverables

## 1.1 Summary of Previous Deliverables

- **First Deliverable (v1.0):** Established the foundation of the ETL project with a focus on basic extraction, transformation, and loading (ETL) of 209,306 traffic accident records (2018-2025) from a CSV file (`traffic_accidents.csv`) into PostgreSQL. Initial transformations (data cleaning, creation of derived columns) and exploratory data analysis (EDA) were performed using Jupyter Notebooks, Pandas, Matplotlib, and Seaborn. Results were visualized through interactive dashboards in Power BI, highlighting temporal, weather, and injury severity patterns.
- **Second Deliverable (v2.0):** Significantly improved the pipeline by introducing orchestration with Apache Airflow, enriching data with geospatial information from OpenStreetMap (OSM), and implementing an optimized dimensional model (`CrashTraffic_Dimensional`) for analytical queries. The repository structure was reorganized, documentation was enhanced, and monitoring and troubleshooting capabilities were added via Airflow.
- **Third Deliverable (v3.0):** Introduces advanced features such as real-time data ingestion with Apache Kafka, data validation with Great Expectations, real-time visualization with Streamlit, and unit testing to ensure code quality. This version consolidates previous functionalities and adds a real-time focus, improving responsiveness to accident events.

**Connection Between Deliverables:**

- **ETL Pipeline Evolution:** The first deliverable established a basic flow (extraction from CSV, simple transformation, loading into PostgreSQL). The second scaled the process with Airflow and a dimensional model, and the third adds real-time streaming with Kafka and advanced validations with Great Expectations.
- **Data Enrichment:** Geospatial data from OSM (second deliverable) is retained, now complemented by real-time data processed via Kafka.
- **Visualization:** The first deliverable used Power BI for historical analysis, the second optimized these dashboards with the dimensional model, and the third adds Streamlit for real-time visualization.

- **Quality and Reliability:** The third deliverable enhances quality with Great Expectations and unit testing, building on the robust structure of previous deliverables.
- **Automation:** Airflow, introduced in the second deliverable, remains the core of orchestration, now managing both batch and real-time workflows.

---

# 2. Project Overview (v3.0)

**Purpose:** Design and implement an advanced ETL pipeline to process traffic accident data (2018-2025), enriched with geospatial and real-time information, to identify patterns, trends, and critical factors for improving road safety.

**Scope:**

- Processing of 209,306 historical records and real-time data.
- Tools: Python 3.12, PostgreSQL 15, Apache Airflow, Apache Kafka, Streamlit, Power BI Desktop, Great Expectations.
- Outcomes: Dimensional database in PostgreSQL, interactive dashboards in Power BI and Streamlit, real-time analysis, and actionable road safety strategies.

**Specific Objectives:**

- Standardize, clean, and enrich data with geospatial information.
- Automate the pipeline with Airflow and process real-time data with Kafka.
- Ensure data quality with Great Expectations and unit tests.
- Detect temporal, weather, severity, and geographic patterns.
- Provide real-time visualizations and historical analysis for road safety decisions.

---

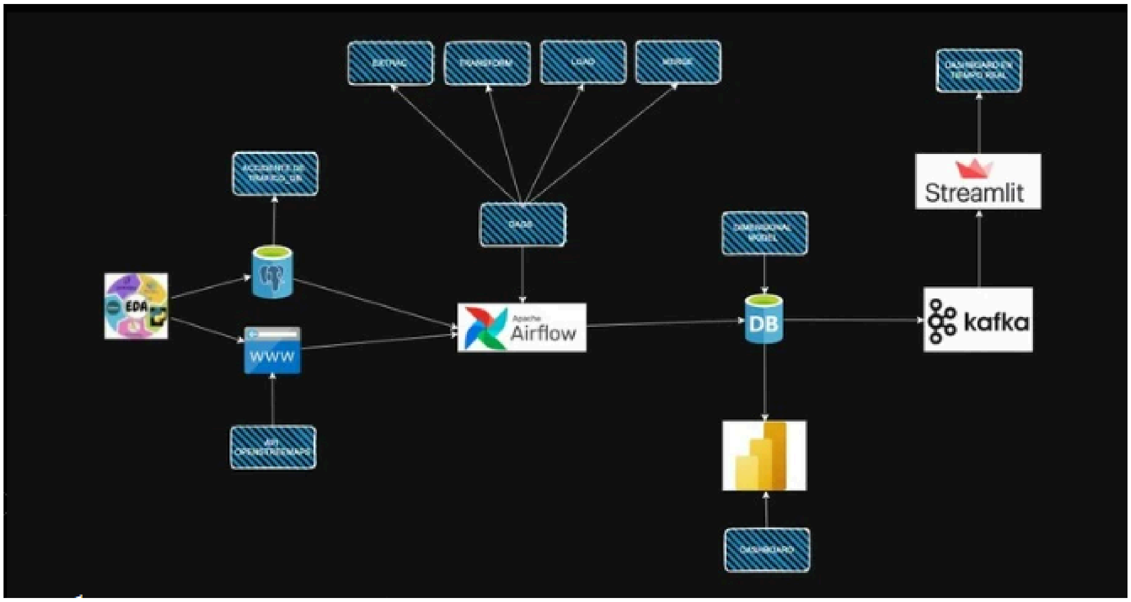
# 3. System Architecture

The current architecture integrates capabilities from previous deliverables with new features for real-time processing and robust validation:

- **Data Sources:**

    - Historical Kaggle dataset (209,306 records, migrated to PostgreSQL: `CrashTraffic`).
    - Geospatial data from OpenStreetMap (OSM).
    - (NEW) Real-time data stream via Kafka.
- **Ingestion and Orchestration:**
    - **Apache Airflow:** Orchestrates both batch (historical ETL) and real-time workflows.
    - **Apache Kafka:** Ingests real-time accident data for instant analysis.
- **Processing and Transformation:**
    - Extraction from PostgreSQL (`CrashTraffic`) and OSM API.
    - Transformations: Cleaning, normalization, geospatial enrichment, creation of derived columns (year, month, day, categories).
    - (NEW) Data validation with **Great Expectations** to ensure completeness, uniqueness, consistency, and validity.
    - Structuring into a dimensional model (star schema) in `CrashTraffic_Dimensional`.
- **Storage:**
    - PostgreSQL: `CrashTraffic` database for raw data and `CrashTraffic_Dimensional` for transformed data.
- **Visualization:**
    - **Power BI Desktop:** Historical dashboards for temporal, weather, and severity trend analysis.
    - **Streamlit:** Real-time dashboard connected to Kafka, displaying accident maps, recent statistics, and hotspots.
- **Code Quality:**
    - (NEW) Unit tests with `pytest` to validate extraction, transformation, and loading functions.

**Architecture Diagram:**




---

# 4. Detailed ETL Process

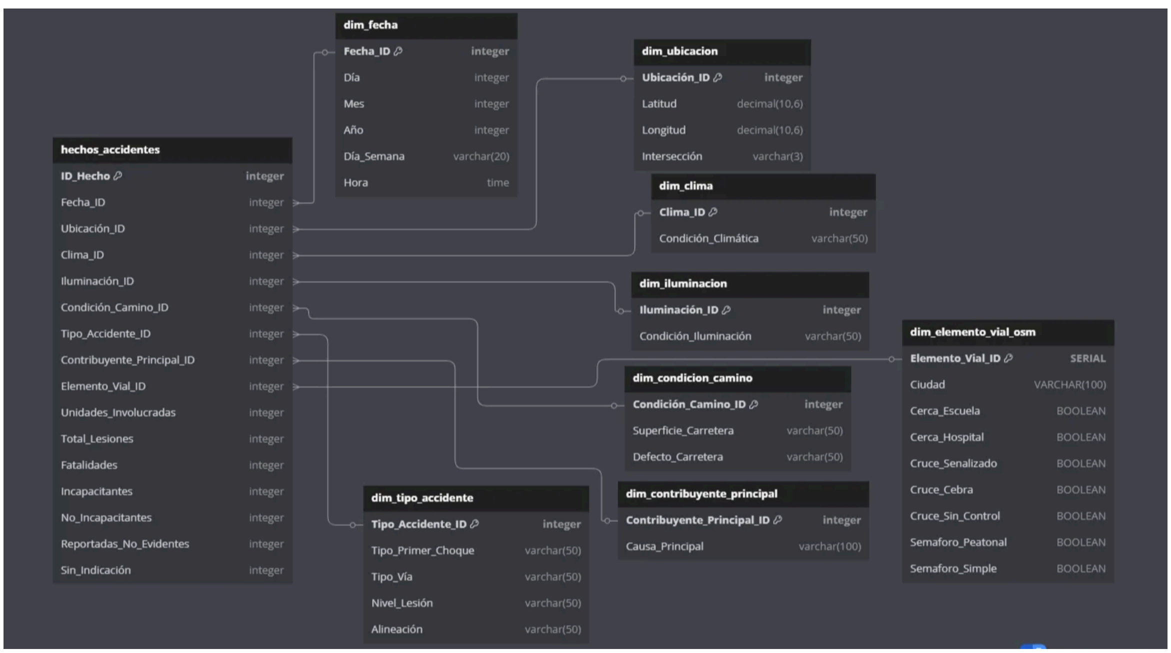
## 4.1 Extraction

- **Sources:**
  - `CrashTraffic` table in PostgreSQL (historical data).
  - OpenStreetMap API (geospatial data).
  - Real-time data stream from Kafka.
- **Methods:**
  - Pandas (`read_csv`, `read_sql`) for historical data.
  - Python scripts for OSM API queries.
  - Kafka consumer for streaming data (`kafka/consumer.py`).

## 4.2 Transformation

- **Cleaning:** Removal of nulls, duplicates, and category standardization.
- **Data Engineering:** Type conversion (dates, coordinates), creation of derived columns (year, month, day, categorical/binary variables).

- **Enrichment:** Integration of geospatial data (e.g., road type, proximity to points of interest).
- **Validation:** Use of Great Expectations to verify:
  - Completeness of critical columns (ID, date, location).
  - Validity of ranges (valid dates, geographic coordinates).
  - Uniqueness of identifiers.
  - Consistency in categories (e.g., weather conditions).
- **Dimensional Model:** Structuring into fact tables (accidents) and dimensions (time, location, weather, severity), with an additional dimension based on OSM API data (`dim_elemento_vial_osm`).



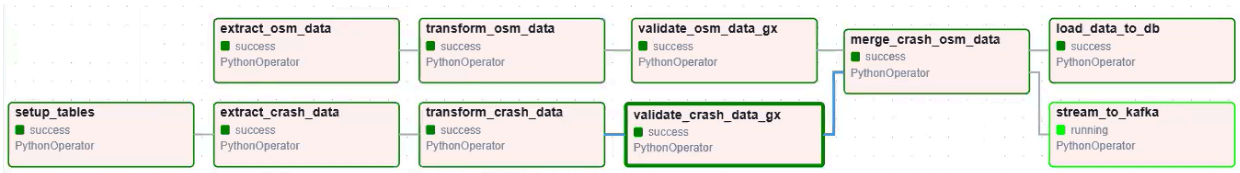

## 4.3 Loading

- **Destination:** `CrashTraffic_Dimensional` database in PostgreSQL.
- **Method:** SQLAlchemy (`to_sql`) via Airflow tasks.

## 4.4 Orchestration

- **Apache Airflow:** DAG `etl_crash_traffic.py` automates pipeline execution, including extraction, transformation, validation, and loading tasks.

- **Monitoring:** Airflow web interface for tracking and troubleshooting.




## 4.5 Streaming

- **Kafka:**
  - Producer (`kafka/producer.py`) sends real-time accident metrics and events.
  - Consumer (`kafka/streamlit_kafka_consumer.py`) processes data for visualization.
- **Streamlit:** Real-time dashboard (`Dashboard/streamlit_app.py`) displaying:
  - Recent accident maps.
  - Real-time statistics (accident count, collision types).
  - High-risk zone (hotspot) identification.

---

# 5. Exploratory Data Analysis (EDA)

- **Notebooks:**
  - `002_EDA_csv.ipynb`: Univariate, bivariate, and temporal analysis of historical data (distributions, correlations, trends).
  - `API_EDA.ipynb`: Analysis of OSM geospatial data (proximity to points of interest, road types).
- **Visualizations:**
  - Matplotlib and Seaborn for static charts (histograms, scatter plots, correlation matrices).
  - Power BI for interactive historical dashboards.
  - Streamlit for dynamic real-time visualization.

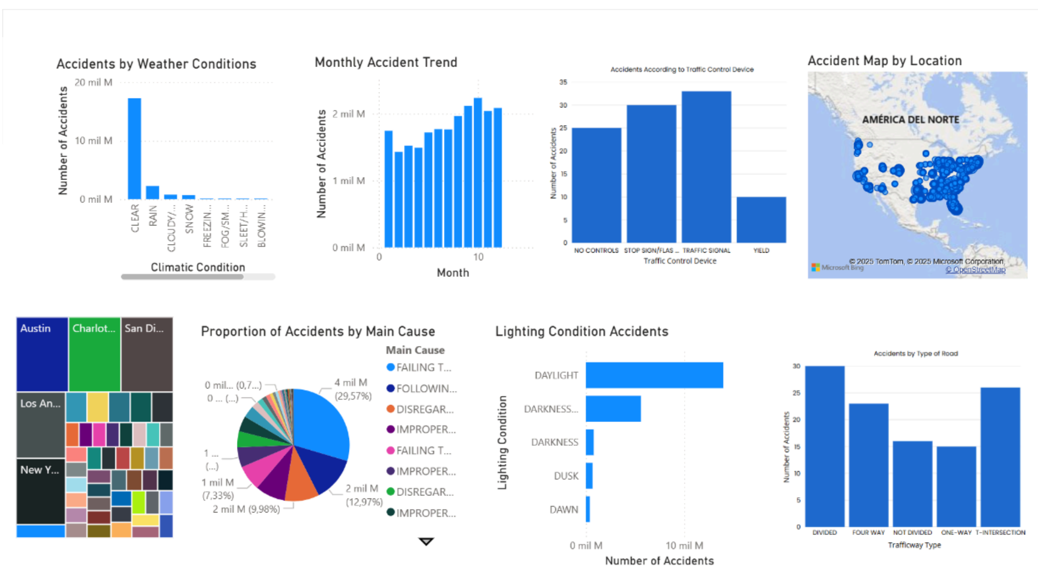
**Key Findings (Consolidated):**

- Accident peaks during rush hours (4:00–5:00 PM) and critical months (January, October).
- Higher accident frequency in clear weather, suggesting human factors.
- Frontal collisions have higher severity but are less frequent.
- Correlations between road infrastructure (intersections, main roads) and accident frequency (enabled by OSM enrichment).

---

# 6. Visualization

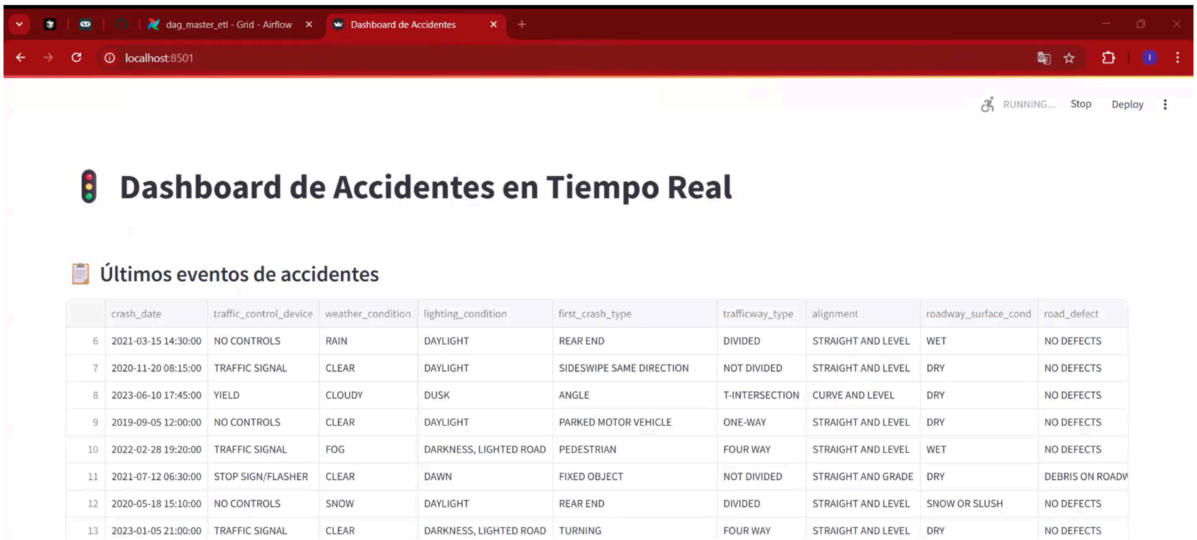
- **Power BI Desktop:**
    - Historical dashboards with:
        - Temporal trends (accidents by hour, day, month).
        - Weather-severity relationships.
        - Collision type distribution.
    - Direct connection to `CrashTraffic_Dimensional`.




- **Streamlit:**
    - Real-time dashboard showing:

- Interactive accident location maps.
- Real-time metrics (accidents per minute, collision types).
- Hotspot identification for immediate response.

- Connected to Kafka via `streamlit_kafka_consumer.py`.




---

# 7. Repository Structure

The repository structure reflects the project’s evolution, integrating previous deliverables with new additions:

```
📂 Traffic-Accidents-ETL
├── API/
│   ├── data/                  # OSM downloaded data
│   ├── notebooks/
│   │   ├── API_EDA.ipynb       # Geospatial data EDA
├── config/
│   ├── conexion_db.py          # PostgreSQL connection configuration
├── dags/
│   ├── etl_crash_traffic.py     # Airflow DAG for ETL pipeline
├── Dashboard/
│   ├── data/
│   │   ├── CrashTraffic.csv     # Original dataset
```

```
│   │   ├── CrashTraffic_clean.csv # Transformed dataset
├── great_expectations/        # Great Expectations configurations
├── kafka/
│   ├── producer.py            # Kafka producer
│   ├── streamlit_kafka_consumer.py # Kafka consumer for Streamlit
├── logs/                     # Airflow and pipeline logs
├── notebooks/
│   ├── 001_extract.ipynb       # Initial extraction and loading
│   ├── 002_EDA_csv.ipynb       # Historical data EDA
├── pdf/                      # PDF documentation
├── tests/
│   ├── test_dag_api_etl.py      # Unit tests for API
│   ├── test_dag_db_etl.py       # Unit tests for database
├── venv/                     # Virtual environment
├── .env                      # Environment variables
├── .gitignore                 # Git-ignored files
├── docker-compose.yaml         # Docker configuration
├── proyecto03.code-workspace    # VS Code configuration
├── README.md                  # Project documentation
├── requirements.txt           # Python dependencies
```

---

# 9. Usage Instructions

1. **Clone the Repository:**

```
git clone https://github.com/isabellaperezcav/Traffic-Accidents-Kafka.git
cd Traffic-Accidents-Kafka
```

2. **Create Virtual Environment:**

```
python -m venv venv
source venv/bin/activate  # Linux/macOS
venv\Scripts\activate     # Windows
```

3. **Install Dependencies:**

```
pip install -r requirements.txt
```

4. **Configure PostgreSQL:**

```
CREATE DATABASE crash_traffic;
CREATE DATABASE crash_traffic_dimensional;
```

Update `config/conexion_db.py` with credentials.

5. **Configure Environment Variables:** Create `.env` with:

```
AIRFLOW__CORE__EXECUTOR=SequentialExecutor
AIRFLOW__DATABASE__SQL_ALCHEMY_CONN=postgresql+psycopg2://user:password@localhost:5432/airflow
DB_HOST=localhost
DB_PORT=5432
DB_USER=your_user
DB_NAME=your_database
DB_PASSWORD=your_password
KAFKA_BOOTSTRAP_SERVERS=localhost:9092
```

6. **Start Services with Docker:**

```
docker-compose up -d
```

Access Airflow at: http://localhost:8080.

7. **Configure Kafka:** Create a topic and verify producer/consumer.

8. **Run Notebooks:**

```
jupyter notebook
```

Execute `001_extract.ipynb`, `002_EDA_csv.ipynb`, and `API_EDA.ipynb`.

9. **Run ETL Pipeline:**

   - Activate the `etl_crash_traffic` DAG in Airflow's web interface.

- Monitor execution and logs.

10. **Run Real-Time Dashboard:**

    ```
    streamlit run Dashboard/streamlit_app.py
    ```

    Access at: http://localhost:8501.

11. **Run Unit Tests (from Docker container):**

    - Access the container:

    ```
    docker-compose exec <app-container-name> /bin/bash
    ```

    ```
    pytest tests/
    ```

    Note: Replace <app-container-name> with the name defined in `docker-compose.yaml`.

---

# 10. Conclusions and Future Improvements

- **Consolidated Findings:**
  - Confirmed patterns: Peaks during rush hours, critical months (January, October), and high severity in frontal collisions.
  - New geospatial insights: Relationships between road infrastructure and accidents.
  - Real-time analysis: Immediate hotspot identification for proactive response.
- **Value of Improvements:**
  - **Automation and Scalability:** Airflow ensures a robust, programmable pipeline.
  - **Real-Time:** Kafka and Streamlit enable instant monitoring.
  - **Data Quality:** Great Expectations ensures reliable data.
  - **Code Reliability:** Unit tests ensure robustness and maintainability.

- **Future Improvements:**
  - Integrate additional data sources (e.g., real-time traffic sensors).
  - Implement predictive models for accident anticipation.
  - Optimize Kafka performance for larger data volumes.
  - Add automated alerts for detected real-time hotspots.

---

## 11. Bibliographic References


1. Apache Software Foundation (n.d.). *Apache Airflow Documentation*. Retrieved from https://airflow.apache.org/docs/
2. PostgreSQL Global Development Group (n.d.). *PostgreSQL Documentation*. Retrieved from https://www.postgresql.org/docs/
3. The SQLAlchemy Project (n.d.). *SQLAlchemy Documentation*. Retrieved from https://docs.sqlalchemy.org/
4. OpenStreetMap Contributors (n.d.). *Overpass API Documentation*. Retrieved from https://wiki.openstreetmap.org/wiki/Overpass_API
5. GeoPy Contributors (n.d.). *GeoPy Documentation*. Retrieved from https://geopy.readthedocs.io/
6. Superconductive (n.d.). *Great Expectations Documentation*. Retrieved from https://docs.greatexpectations.io/docs/home/
7. Apache Software Foundation (n.d.). *Apache Kafka Documentation*. Retrieved from https://kafka.apache.org/documentation/
8. Docker Inc. (n.d.). *Docker Documentation*. Retrieved from https://docs.docker.com/
9. Microsoft Corporation (n.d.). *Power BI Documentation*. Retrieved from https://learn.microsoft.com/en-us/power-bi/