Data Analytics Specialist Test

Cleaning, Validation, and Analysis of a Historical Aviation Crashes

Jorge A. Garcia

February 23, 2026

Outline

1. About me
2. General approach
3. Project structure
4. Data Cleaning
5. Data Validation
6. Metadata
7. Analysis
8. User Interface
9. Summary
10. Disclosure

About me

- Quantitative modeler with 10+ years of experience:
  - Optimization (e.g., LP, ML)
  - Agent-based simulation
  - Physics-based modeling
  - Macroeconomic models
- Currently, developer of web-apps for science and decision-making (https://modelbridgelabs.com/).
- Goals:
  - Apply data analytics and modeling to inform decision-making and uncover actionable insights.
  - Learn and grow as a professional by taking on new challenges and responsibilities.

General approach

1. Exploratory field analysis
2. Structure design
3. Data cleaning
4. Automated validation
5. Data profiling
6. Version control, CI, testing (github.com/jorge-antares/das_test)
7. Iterative refinement

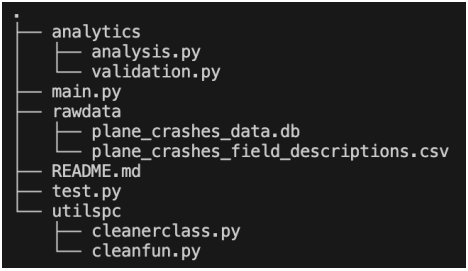

Figure: Project structure.

Project structure

- `utilspc/`
  - `cleanfun.py` — Parsing functions for cleaning specific fields.
  - `cleanerclass.py` — Python classes for SQLite connection and cleaning.
- `analytics/`
  - `validation.py` — validation checks and profiling report generation.
  - `analysis.py` — analysis functions for generating reports and insights.
- `main.py` — main script to run the cleaning, validation, and analysis pipeline.
- `test.py` — simple validity test (CI - GitHub Actions).
- `output`
  - `metadata.csv` — column-level profile of the cleaned dataset.
  - `data_profile.txt` — full analysis report of the cleaned dataset.
  - `cleaned_plane_crashes.db` — cleaned SQLite database.
  - `validation_report.txt` — detailed validation report with check results and failure details.

Date, Time and Location

- `date`: Ambiguity in YY-MM-DD or DD-MM-YY formats is resolved by assuming the former. Parse 'DD-Mon-YY' to 'YYYY-MM-DD', or None/Null on failure. UTF?
- `time`: Normalise to 'HH:MM' (24-hour). Strips prefix 'c' with optional space. Handles 4-digit integers (e.g. '1730' to '17:30'). Returns None for unparseable values.
- `location` Returns a tuple ('first', 'last') where 'first' is the location excluding the country (or None) and 'last' is the country name (or None). Creates field `country`.

Operator, AC type, aboard and fatalities

- `operator`: Strip whitespace, map '?' to NULL and nullify values that are clearly not airline operators:
  - Pure serial / registration codes, e.g. "'46826/109' ".
  - Aircraft manufacturer + model designator entries that were incorrectly placed in the operator column, e.g. 'Boeing KC-135E', 'Lockheed AC-130H Hercules'.
- `ac_type`:
  - Strip whitespace and map '?' to NULL.
  - Remove vehicle categories, e.g. '(flying boat)', '(airship)', '(amphibian)'.
  - Remove extra spaces.
  - Strip any residual leading '/' trailing whitespace.
  - Return None is empty
- `aboard, fatalities`: Returns (total, passengers, crew) as int or None.

Data Validation

Automated checks run after every cleaning pass:

Structure

- Schema: all 18 expected columns with correct declared types
- Python-level type consistency per column

Format

- Dates match `YYYY-MM-DD`; range 1908–2025
- Times match `HH:MM` (24-hour)
- All numeric columns ≥ 0

Cross-column consistency

- `aboard_passengers + aboard_crew = aboard_total`
- `fatalities_passengers + fatalities_crew = fatalities_aboard`
- `fatalities_aboard` $\leq$ `aboard_total`
- `fatalities_total = fatalities_aboard + ground`

Duplicates

- Rows sharing `(date, operator, route)` flagged as potential duplicates

Results reported as **PASS / WARN / FAIL** per check.

Dataset Schema

metadata.csv (5,783 rows)

Field	Type	NA	Unique	Description
date	TEXT	0	5212	Date of the crash (YYYY-MM-DD)
time	TEXT	2183	1051	Time of the crash (HH:MM, 24-hour)
location	TEXT	230	4372	Location of the crash
country	TEXT	104	429	Country of crash (derived from location)
operator	TEXT	29	2812	Airline or operator
flight_no	TEXT	4508	868	Flight number assigned by the operator
route	TEXT	1689	3839	Route flown prior to the accident
ac_type	TEXT	24	2666	Aircraft type
registration	TEXT	357	5385	ICAO registration of the aircraft
cn_ln	TEXT	1209	4063	Construction/serial number
aboard_total	INTEGER	40	242	Total number of people aboard
aboard_passengers	INTEGER	543	231	Number of passengers aboard
aboard_crew	INTEGER	539	31	Number of crew aboard
fatalities_aboard	INTEGER	11	200	Total number of fatalities aboard
fatalities_passengers	INTEGER	558	190	Number of passenger fatalities
fatalities_crew	INTEGER	556	28	Number of crew fatalities
ground	INTEGER	52	53	Ground fatalities
fatalities_total	INTEGER	0	204	Total number of fatalities
summary	TEXT	231	5366	Brief description of the accident

Analysis

Five sections produced in the profiling report:

1. **Data Profiling** — NULL rate, unique value count, and type per column.
2. **Descriptive Statistics** — Aggregate fatality rate (fatalities / aboard), survival rate, ground casualties, and crew vs. passenger fatality split.
3. **Trend Analysis** — Crashes and fatalities per decade and per year (ASCII bar chart); top 15 operators by crash count; most dangerous aircraft types by fatality rate (min. 10 incidents).
4. **Geographic Analysis** — Top 20 countries/regions and specific crash sites by incident count.
5. **Data Quality** — Mismatched totals, rows where fatalities exceed aboard count, duplicate (`date, operator, route`) groups, and registration reuse across aircraft types.

User Interface

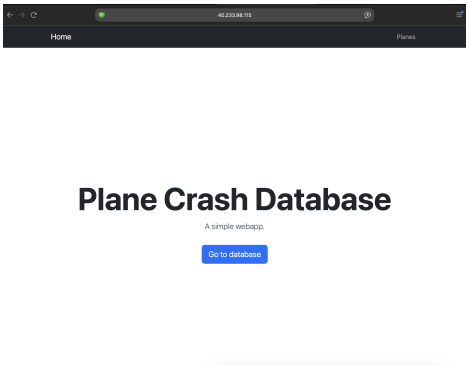

Figure: Web app interface at https://40.233.98.115/.

Frameworks used:

- Flask for backend API.
- Jinja2 for HTML templating.
- Bootstrap for responsive design.
- Docker for containerisation and deployment.
- Nginx for reverse proxy and static file serving.

Hosting via a VM instance (Ubuntu server 24.04.4 LTS) on Oracle Cloud.

Summary

My approach:

- Develop a modular and reusable structure with minimal external dependencies.
- Implement programming best practices: version control, testing, CI/CD, and documentation.
- Standardise fields to keep consistency and enable analysis.
- Cross-validate data and create metrics (e.g., fatality rates).
- Explore ways to integrate data products with decision-making processes and end-user applications.

Disclosure of Tools

Claude Sonnet 4.6 was used in the following tasks:

- Parse fields to look for edge cases.
- Create documentation based on project structure and code comments.
- Create HTML for user interface.
- Code completion and debugging suggestions (VS Code).

All LLM outputs were reviewed and edited.