Data Analytics Specialist Test

Cleaning, Validation, and Analysis of a Historical Aviation Crashes

Jorge A. Garcia

February 23, 2026

Outline

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 analysis
2. Structure design
3. Data cleaning
4. Automated validation
5. Data profiling
6. Version control, CI, testing
7. Iterative refinement

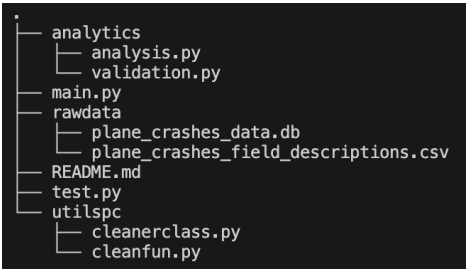

Figure: Project structure.

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` — test for type checking, cleaning functions, and validation checks.
- `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`: Parse 'DD-Mon-YY' to 'YYYY-MM-DD', or None/Null on failure. Year values ¡ 26 are treated as 2000s, otherwise 1900s.
- `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 1908–2018
- Times match `HH:MM` (24-hour)
- All numeric columns ≥ 0

Cross-column consistency

- `aboard_pax + aboard_crew = aboard_total`
- `fat_pax + fat_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.

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.

Summary and Future Work

My approach:

- Develop a modular and reusable structure.
- Implement programming best practices: version control, testing, CI/CD, and documentation.
- Explore ways to integrate data products with decision-making processes and end-user applications.

Improvements:

- Enhance location parsing to better handle edge cases and extract more granular geographic information.
- Implement additional validation checks for outliers and temporal inconsistencies (e.g., crashes dated before the Wright brothers' first flight).
- Standardize operator names, AC types, and geographic scales.
- Complement data with external reference datasets (e.g., GIS, weather, economic indicators).