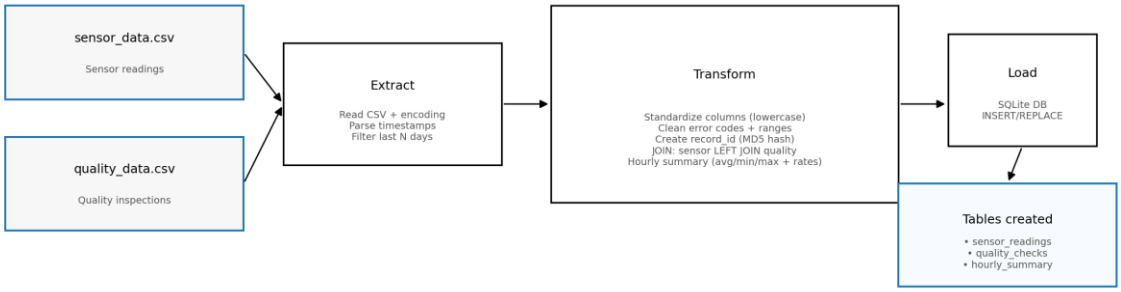
# ETL Pipeline Report

### Sensor Readings + Quality Inspections → SQLite Database

Main purpose is to show the ETL data flow, describe the key transformations (especially the join), and provide sample SQL queries with database outputs.

## Section 1 - Data Flow Diagram

Inputs (2 CSV files) → Extract → Transform (clean + join + aggregate) → Load into SQLite.




## Section 2 - Transformation Logic :

**Standardize:** lower-case column names; normalize identifiers; create a stable record_id using an MD5 hash of key fields.

**Clean:** convert error codes (-999, -1) and NULL-like values to NaN; validate ranges (T 0-150, P 0-10, V 0-100); forward-fill per machine; label rows as good, estimated, or invalid depending on missingness before/after fill.

**Join operation (left join):**

1) Select join keys in priority order: (timestamp, line_id, machine_id) else (timestamp, machine_id).

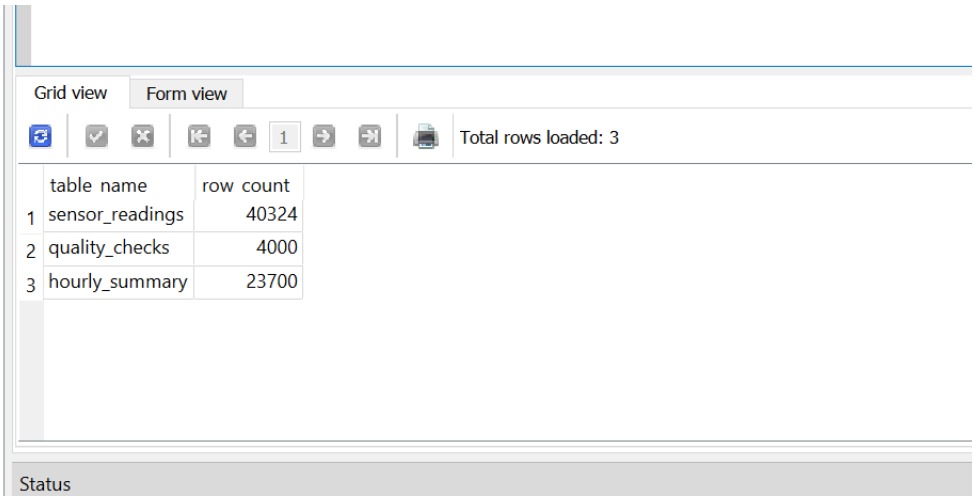
2) De-duplicate the quality dataset per join key (keep last) to avoid multiplying sensor rows.

3) Merge quality fields into sensor rows. Compute quality_status as passed / failed / not_checked (missing quality).

## Section 3 - Sample Queries and Results

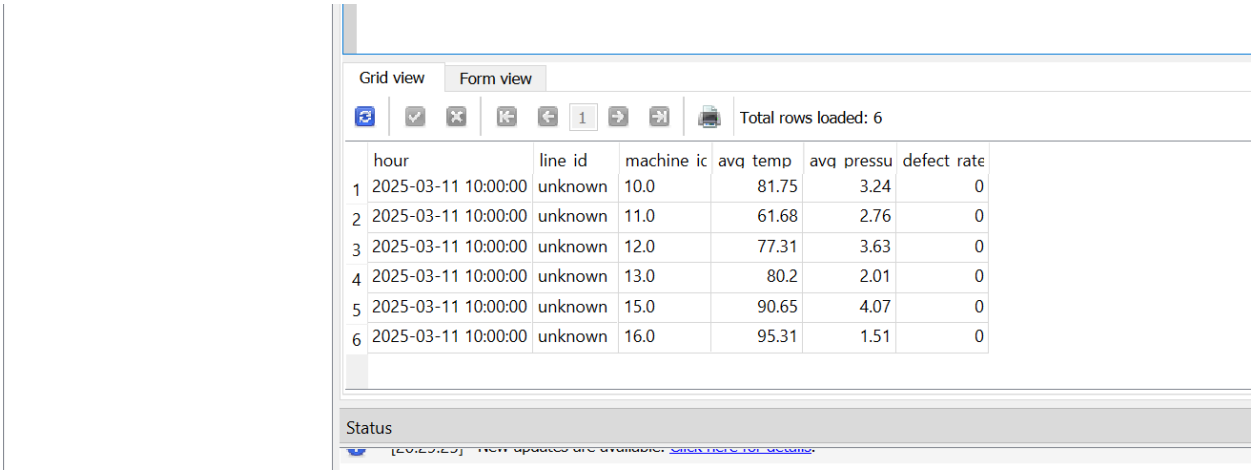
### SQL Query

```
-- Q1) Verify loaded data volume
SELECT 'sensor_readings' AS table_name, COUNT(*) AS row_count FROM sensor_readings
UNION ALL SELECT 'quality_checks', COUNT(*) FROM quality_checks
UNION ALL SELECT 'hourly_summary', COUNT(*) FROM hourly_summary;
```



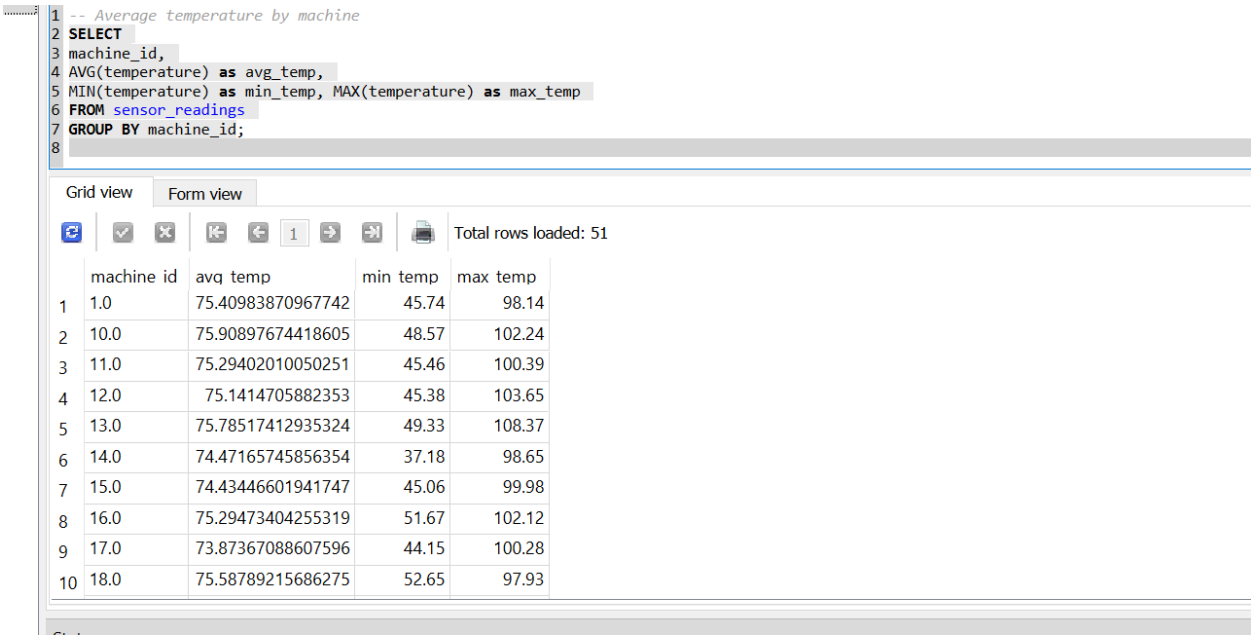

### SQL Query

```
-- Q2) Find hours/machines with the highest defect rates
SELECT hour, line_id, machine_id,
       ROUND(avg_temperature,2) AS avg_temp,
       ROUND(avg_pressure,2) AS avg_pressure,
       ROUND(defect_rate,1) AS defect_rate_pct
FROM hourly_summary
ORDER BY defect_rate_pct DESC, hour DESC
LIMIT 6;
```



**SQL Query**

```
-- Average temperature by machine
SELECT
       machine_id,
       AVG(temperature) as avg_temp,
       MIN(temperature) as min_temp, MAX(temperature) as max_temp
FROM sensor_readings
GROUP BY machine_id;
```



```
1 -- Average temperature by machine
2 SELECT
3 machine_id,
4 AVG(temperature) as avg_temp,
5 MIN(temperature) as min_temp, MAX(temperature) as max_temp
6 FROM sensor_readings
7 GROUP BY machine_id;
8
```

Grid view | Form view

Total rows loaded: 51

| | machine id | avg temp | min temp | max temp |
|---|---|---|---|---|
| 1 | 1.0 | 75.40983870967742 | 45.74 | 98.14 |
| 2 | 10.0 | 75.90897674418605 | 48.57 | 102.24 |
| 3 | 11.0 | 75.29402010050251 | 45.46 | 100.39 |
| 4 | 12.0 | 75.1414705882353 | 45.38 | 103.65 |
| 5 | 13.0 | 75.78517412935324 | 49.33 | 108.37 |
| 6 | 14.0 | 74.47165745856354 | 37.18 | 98.65 |
| 7 | 15.0 | 74.43446601941747 | 45.06 | 99.98 |
| 8 | 16.0 | 75.29473404255319 | 51.67 | 102.12 |
| 9 | 17.0 | 73.87367088607596 | 44.15 | 100.28 |
| 10 | 18.0 | 75.58789215686275 | 52.65 | 97.93 |

## Section 4 - Challenges and Solutions

- **Inconsistent column names** (Timestamp vs timestamp) → standardize headers before processing.
- **Encoding / delimiter issues** → implement read fallbacks (UTF-8 then ISO-8859-1; ';' separator fallback).
- **Join duplicates** when multiple quality rows match one sensor key → de-duplicate quality per join key before merging.
- **Time misalignment** (checks not at exact sensor timestamp) → use a nearest-timestamp (time-window) join if needed.
- **Invalid sensor values** (-999, out-of-range) → replace with NaN, validate ranges, forward-fill cautiously, and track data_quality.

**Reference:** Python ETL script (student implementation) defining extraction, cleaning, standardization, join, aggregation, and SQLite loading.