Perceive Now – Data Validation & Schema Drift Assessment

Candidate: Madhu Bala

Role: Data Engineer – Validation & Quality

Story 1 — The Case of the Contradicting Sources

1. How I Define a Contradiction Between Two Structured Datasets

A contradiction occurs when two authoritative sources provide different values for the same entity and field **beyond an acceptable tolerance.** The definition varies depending on data type:

Numeric fields

A contradiction exists if:

```
relative_difference = |A - B| / max(|A|, |B|) > threshold (e.g., 5%)
```

Example:
Salesforce revenue = 100000
HubSpot revenue = 99000
Relative diff = 1% → acceptable
Relative diff = 10% → contradiction

String fields

Use fuzzy similarity or cosine similarity:

```
similarity(name_a, name_b) < 0.85 → contradiction
```

Example: “Acme Corp” vs “Acme Corporation”

Null vs non-null

If one source reports a value and the other doesn’t → **missing_one** inconsistency.

Timestamp comparison

When both disagree, but one value is **newer and from a more reliable source**, that increases its weight.

---

2. Detecting Inconsistencies Across Sources

My detection framework performs:

Step 1: Canonical Join

Outer join on the canonical key (e.g., `customer_id`).
This ensures we include matches, mismatches, and records appearing in only one dataset.

Step 2: Field-Level Checks

For each overlapping field:

Field Type	Validation Method
Numeric	Relative delta comparison
Text	Fuzzy ratio (SequenceMatcher)
Timestamp	Recency comparison
Missing values	Null analysis

Step 3: Record-Level Classification

A record is marked:

- `match`
- `missing_one`
- `contradiction`
- `missing_both`

---

3. Confidence Scoring (How I Score Each Record)

I assign a confidence score between **0 and 1** based on:

(A) Source Reliability

Example scores:

- Salesforce: 0.95
- HubSpot: 0.90

(B) Recency Score

Fresher data → higher score
Older data → penalized

(C) Conflict Frequency

If a customer is frequently contradictory, confidence decreases.

(D) Agreement Across Sources

If both sources match → +confidence
If they contradict → -confidence

Formula Used in My Script

```
confidence = (0.4 * reliability)
           + (0.25 * recency)
           + (0.25 * agreement)
```

```
    - (0.2 * conflict_frequency)
```

All values are scaled into a 0–1 range.

---

4. Demo Code Summary (Contradiction Detector)

The script does:

- Loads two CSV files (`salesforce.csv` and `hubspot.csv`)
- Compares values field-by-field
- Identifies contradictions using numeric delta and fuzzy logic
- Computes confidence scores
- Generates:

Output files created:

- `validation_summary.csv`
- `validation.db` (SQLite)
- Visualization of confidence scores

Story 2 — The Mystery of the Missing Fields

1. How I Implement Schema Drift Detection

Step 1: Schema Snapshot

For each dataset version, I capture:

- Field names
- Inferred type (`int`, `float`, `string`, `datetime`)
- Nullability
- Cardinality sample

Step 2: Compare Snapshots

I detect:

Drift Type	Description
`new_fields`	Fields added in newer dataset
`removed_fields`	Fields missing in newer dataset
`type_mismatches`	Datatype changed (e.g., float → string)

Step 3: Drift Report Output

Example output:

```
{
  "new_fields": ["region_code"],
  "removed_fields": ["industry_group"],
  "type_mismatches": {
    "revenue": ["float", "string"]
  }
}
```

Step 4: Persist to DB

Results stored in `schema_drift.db` for future auditing.

---

2. How I Track Field-Level Lineage

I maintain:

- A `schema_registry` table storing all historical schema snapshots
- A `drift_events` table storing every detected drift
- Timestamps + version IDs for reconstructing lineage
- Ability to trace when, who, and how a field changed over time

---

3. What I Alert and Visualize

Alerts:

- Missing critical field → **immediate alert**
- Type mismatch in numeric field → high-severity
- Large schema changes (>10%) → warning

Visualization:

- Grafana dashboard showing:
    - Drift frequency
    - Field change history
    - Schema version timelines

---

4. Proof-of-Concept Script

My script:

- Reads two CSV versions
- Computes schema metadata
- Detects field drift
- Outputs a JSON file
- Stores drift events in SQLite

Files generated:

- `drift_report.json`
- `schema_drift.db`

Final Deliverables Provided

✓ `contradiction_detector.py`
✓ `schema_drift_detector.py`
✓ `salesforce.csv`
✓ `hubspot.csv`
✓ `validation_summary.csv`
✓ `validation.db`
✓ `drift_report.json`
✓ `schema_drift.db`
✓ This document (convert to PDF)

How to Run My Code (for reviewer)

Contradiction Detector

```
python contradiction_detector.py --a salesforce.csv --b hubspot.csv
--run_once
```

Schema Drift Detector

```
python schema_drift_detector.py --v1 salesforce.csv --v2 hubspot.csv
--out drift_report.json
```

Closing Note

My solution focuses on:

- Deterministic validation
- Quantifying uncertainty
- Accurate contradiction detection
- Schema drift detection
- Auditability and lineage
- Scalable design for 100+ data sources
- Practical implementation using Pandas, NumPy, SQLite