The Sample report created based on the test data (good_data.csv and bad_data.csv)

Summary

- I attempted an automated quick validation but that ran into an execution error, so I performed a manual audit of the two CSVs you provided against the two rules:
    1. user_id must not be null
    2. sale_amount must be a positive number

Files reviewed

1. good_data.csv (3 data rows) Header: user_id,sale_amount,item_name Data rows:

- Row 1: 101,50.00,Widget A
- Row 2: 102,120.50,Widget B
- Row 3: 103,75.25,Widget C

Result: All checks pass.

- user_id: all present
- sale_amount: all positive numeric values Actionable result: This file is valid for ingestion under the given rules.

2. bad_data.csv (4 data rows) Header: user_id,sale_amount,item_name Data rows:

- Row 1: 101,50.00,Widget A — OK
- Row 2: ,120.50,Widget B — FAIL: user_id is NULL
- Row 3: 103,-25.00,Widget C — FAIL: sale_amount is negative (-25.00)
- Row 4: 104,15.00,Widget D — OK

Result: 2 failing rows (50% of this file’s rows is fine, 50% have issues)

- Rule 1 (user_id not null): 1 failure — row 2 (empty user_id)
- Rule 2 (sale_amount positive): 1 failure — row 3 (sale_amount = -25.00)

Detailed recommendations (immediate remediation)

1. Quarantine the failing rows

    - Move rows 2 and 3 from bad_data.csv to a quarantine/error table or file with metadata (source file, line number, timestamp, rule violated).
2. Reject or hold ingestion of quarantined rows until corrected or approved.
    - If automatic fix is possible (e.g., user_id can be looked up), apply that only after validation and logging.
3. For the negative sale_amount (row 3):
    - Verify whether negative amounts represent valid business events (refunds/returns). If they should be allowed, update the business rule and downstream consumers. If not, treat as data error and request correction from the source.
4. For the missing user_id (row 2):
    - Attempt enrichment (lookup by transaction id or other source metadata). If impossible, request corrected source file / backfill.

Suggested validations to implement in the pipeline

- Pre-ingest:
    - Schema validation: enforce user_id as NOT NULL, sale_amount numeric (and > 0 if appropriate).
    - Row-level policy: route failures to quarantine with reason codes.
- Ingest-time checks:
    - Block ingestion if failure rate > threshold (e.g., >1% or customize per source).
    - For low failure rates, allow partial ingest but send alerts and quarantine failing rows.
- Post-ingest:
    - Data quality dashboard with metrics: row counts, null counts, negative amounts, failure rate per source, daily trend.
    - Alerts: immediate alert (Slack/email) when a source exceeds thresholds or critical fields contain nulls/invalid values.

Examples: queries and checks you can run

- SQL to find failing rows:
    - Missing user_id: SELECT * FROM staging_table WHERE user_id IS NULL;
    - Negative sale_amount: SELECT * FROM staging_table WHERE TRY_CAST(sale_amount AS NUMERIC) IS NULL OR CAST(sale_amount AS NUMERIC) <= 0;
- Python / pandas quick checks:
    - df['user_id'].isnull().sum()
    - (pd.to_numeric(df['sale_amount'], errors='coerce') <= 0).sum()

Monitoring and tooling recommendations

- Use a data quality framework:
  - Great Expectations, Deequ, or custom validators to encode rules and run them automatically.
- Observability:
  - Capture metrics per-file and per-source: total rows, failed rows, failure rate, last successful ingest time.
  - Create alerts for spikes in failure rate or repeating failing patterns.
- Data contract & SLA:
  - Define a data contract with upstream producers that includes required fields, allowed value ranges, and expected failure rate.
  - Enforce contract with CI checks for data producers where possible.

Business-level suggestions

- Define policy for negative sale_amounts (allowed vs disallowed).
- Define remediation SLAs for quarantined rows (e.g., source must correct within 24 hours).
- Add lineage metadata so you can trace back each row to its source and file.

If you want, I can:

- Produce a small shell script / SQL / Great Expectations suite that implements the two rules and quarantines failing rows.
- Generate a sample quarantine/error report with recommended columns (source_file, line_number, raw_row, failure_reasons, timestamp).
- Help design alert thresholds and an initial dashboard.

Which follow-up would you like? (Generate validation code, quarantine report template, or implementation checklist?)