

# Computational Skepticism for Data Understanding: A Repeatable QC System (Detect–Fix–Communicate)

---

This chapter proposes a reusable QC system that operationalizes **computational skepticism** through a **Detect–Fix–Communicate** workflow. The system standardizes core validation checks across datasets and extends to domain rules via a plugin design.

---

## 1. Introduction

### Research Question

**How can we design a general, repeatable QC system—grounded in computational skepticism—that detects common data failures, applies auditable fixes, and communicates trustworthiness clearly, while supporting domain-specific validation rules as modular plugins?**

### Relevance and Interest

Real-world data validation is often **ad hoc**: scattered checks in notebooks, one-off SQL queries, and “tribal knowledge.” This slows analysis, reduces reproducibility, and increases the risk that decisions are driven by **data artifacts** (e.g., duplicates, missingness patterns, invalid values) rather than true signal. A general system cannot capture every domain nuance, but it can standardize the **common failure modes** and then allow domain-specific checks to be added as plugins—improving efficiency and reliability while preserving flexibility.

---

## 2. Background and Theory

### What does “data understanding” mean in practice?

In this course, understanding data is not limited to reading columns and computing summary statistics. It includes:

- **Trustworthiness:** Can the data support the intended inference or decision?
- **Failure modes:** What might be wrong (missingness, outliers, errors, drift)?
- **Impact:** How do preprocessing and validation change conclusions?
- **Communication:** How do we present issues and improvements clearly and honestly?

### Computational skepticism

**Computational skepticism** is an algorithmic mindset:

*Treat the dataset as untrusted until it passes evidence-based checks.*

Instead of assuming the dataset is “clean enough,” computational skepticism:

1. Makes assumptions explicit (schema, constraints, expected ranges/distributions).
2. Tests assumptions with systematic checks.

3. Prioritizes issues with severity scoring.
4. Applies **auditable** fixes (reproducible and documented).
5. Communicates remaining uncertainty clearly.

This aligns with the course arc:

- **Week 4:** preprocessing + validation + critical evaluation
- **Week 5:** data improvement + computational skepticism + chart selection
- **Week 6:** visual design + effective communication

Why a general system plus plugins?

Each dataset is unique, but data issues repeat across domains. A practical strategy is:

- **QC Core:** General checks that apply broadly (types, missingness, duplicates, outliers, constraint violations).
- **Domain Plugins:** Additional rules and derived checks that depend on domain semantics.

This mirrors validation systems used in practice, including expectation-based validation approaches (where data must satisfy a set of checks/expectations).

Supporting foundations and related work

- **Tidy structure** reduces ambiguity: each variable is a column, each observation is a row, and each observational unit forms a table. This makes validation and downstream analysis more reliable.
- **Data validation in pipelines** is recognized as critical for reliable downstream ML/analytics and monitoring quality over time.
- **Effective visual communication** helps QC findings become actionable rather than ignored (e.g., chart choice, annotation, hierarchy, and reducing clutter).

(See references in Section 7.)

---

### 3. Problem Definition

Goal

Design a **repeatable QC pipeline** that:

1. **Detects** common data problems through skeptical checks,
2. **Fixes** them using reproducible, auditable transformations,
3. **Communicates** data trustworthiness and improvement using clear visuals,
4. Supports **domain-specific validation** through plugins (e.g., a claims plugin or a housing plugin).

Input–Output Definition

**Input**

- Dataset **D** (tabular data)
- Metadata **M** describing expectations:
  - schema (types, required fields)

- constraints (single-field and cross-field rules)
- group keys (for subgroup checks)
- optional time column (for drift checks)
- Optional plugin P (domain-specific checks + derived validations)

## Output

- D\_clean: cleaned dataset + **quality flags** (not silent deletion)
- QCReport: failed checks + evidence + severity + recommended action
- VisualSummary: plots demonstrating issues and improvements (before/after)
- QCScore: interpretable trust score aggregated from check results

Sample Data (Healthcare claims-style example)

### Input data (sample rows)

ClaimId	HCP_ID	NDC	DaysSupply	Qty	FillDate	PaidAmount	PlanType
101	9001	A12	30	30	2024-01-10	120.50	COMM
102	9001	A12	-10	30	2024-01-15	118.00	COMM
103	9120	B34	90	90	2024-02-02	-25.00	MEDD
104	9120	B34	(missing)	90	2024-02-20	260.00	COMM

### Example outputs (high level)

- Constraint violation detected: DaysSupply <= 0
- Constraint violation detected: PaidAmount < 0
- Missingness detected: DaysSupply missing; missingness by subgroup (e.g., PlanType) reported
- Fix recommendations:
  - set invalid DaysSupply to missing + add DaysSupply\_invalid flag (high severity)
  - set negative PaidAmount to missing or exclude depending on business rule + log (high severity)
  - impute DaysSupply (median by NDC or therapeutic class) + add missingness flag (medium severity)

## 4. Assumptions and Analysis

### Constraints and assumptions

- **Skew and heavy tails:** Claims amounts and utilization measures are often highly skewed; naive z-scores can over-flag. Use robust methods (IQR/MAD) and anomaly scoring rather than binary "outlier vs not."
- **Missingness is structured:** Missing values may correlate with payer type, channel, geography, or data vendor. Validation must check missingness **by subgroup**, not only globally.
- **Outliers may be valid:** Some HCPs legitimately prescribe at extreme volumes; QC should prefer **flagging + explanation** over automatic removal.

- **Duplicates are not always trivial:** Duplicate claims can occur due to resubmissions or vendor joins. Deduplication rules must be deterministic and documented.
- **Fixes must be auditable:** Each change must be reproducible and recorded (what changed, why, and impact).

## Logic and approach (key algorithmic principles)

This QC system applies core “data understanding” principles:

- **Critical evaluation before analysis:** validate assumptions before deriving conclusions.
  - **Robust statistics:** resist distortion by outliers and skew.
  - **Constraint reasoning:** encode “impossible states” as checks.
  - **Separation of concerns:** detect vs fix vs communicate are distinct stages.
  - **Modularity:** core checks are reusable; domain rules are plugins.
- 

## 5. Proposed System: Detect → Fix → Communicate

### Overview

The system is a pipeline with a **QC Core** plus optional **Domain Plugin(s)**.

#### A) Detect (skeptical checks)

##### **QC Core modules (dataset-agnostic)**

1. **Schema & type checks:** parsing/type mismatches, invalid categories
2. **Missingness analysis:** % missing per column + missingness by subgroup
3. **Anomaly scoring:** robust outlier detection (IQR/MAD) + distribution-aware ranking
4. **Constraints:** single-field (e.g., non-negative amounts) and cross-field rules
5. **Duplicate/integrity checks:** duplicates on key fields, conflicting duplicates
6. **(Optional) Drift checks:** if time exists, compare distributions across windows

##### **Claims plugin adds**

- domain constraints (e.g., `DaysSupply > 0, Qty > 0, PaidAmount >= 0`)
- derived validations and engineered features:
  - `DailyDoseProxy = Qty / DaysSupply` (when meaningful)
  - `Utilization_30d` per HCP and NDC
  - `HighPrescriberFlag` using robust thresholds within specialty/region
- domain-specific severity adjustments (e.g., missing `HCP_ID` or `FillDate` might be high severity)

##### **Housing plugin adds**

- domain constraints (e.g., `SalePrice > 0, GrLivArea > 0`, plausible `YearBuilt`)
- derived validations (e.g., `PricePerSqFt = SalePrice / GrLivArea`)
- domain-specific severity adjustments (e.g., missing neighborhood might be high severity)

#### B) Fix (auditable transformations)

Fixes are conservative and explainable:

- **Missing values:** impute (median/mode) + add missingness flag; or preserve missing if meaningful
- **Outliers:** cap/winsorize or flag as special-case (avoid silent deletion)
- **Invalid values:** convert impossible values to missing + flag, or exclude only with explicit constraint justification
- **Duplicates:** deduplicate using deterministic rule (e.g., keep most complete record)

All fixes produce a **change log**: what changed, how many rows were affected, and why.

### C) Communicate (QC report + visuals)

The pipeline generates a QC report and visual summaries:

- Top failing checks by severity
- Before/after plots showing improvement
- Notes on remaining uncertainty and assumptions
- A trust score breakdown (so **QCScore** is explainable)

### Pseudocode

```
function QC_PIPELINE(D, metadata M, plugin P=None):
    report = []

    # 1) DETECT (QC Core)
    report += schema_checks(D, M.schema)
    report += missingness_checks(D, group_keys=M.group_keys)
    report += anomaly_scoring(D, robust=True)
    report += constraint_checks(D, M.constraints)
    report += duplicate_checks(D, key=M.primary_key)

    if M.time_column exists:
        report += drift_checks(D, time=M.time_column)

    # Domain-specific DETECT
    if P is not None:
        report += P.detect(D, M)

    # 2) FIX (auditable)
    fixes = propose_fixes(report, policy="conservative_with_flags")
    D_clean, fix_log = apply_fixes(D, fixes)

    # 3) COMMUNICATE (visual + report)
    visuals = generate_visual_summary(D, D_clean, report)
    qc_score = aggregate_score(report)

    return D_clean, report, fix_log, visuals, qc_score
```

### Proof sketch (reasoning / correctness)

- Schema and constraint checks reduce logically impossible states and parsing errors that can invalidate downstream metrics.

- Missingness analysis detects both global and subgroup-specific incompleteness that can bias comparisons.
  - Robust anomaly scoring reduces sensitivity to heavy tails and identifies suspicious values without assuming normality.
  - Deterministic fix rules improve reproducibility and avoid “silent analyst choices.”
  - Communication artifacts prevent QC from becoming invisible and ensure users understand what changed and what uncertainty remains.
- 

## 6. Results and Discussion

**Note:** The final numeric results and figures are produced in [Analysis.ipynb](#). This section summarizes what will be reported and how it connects to theory.

### Dataset 1: Healthcare Claims (QC Core + Claims Plugin)

#### Expected findings

- Constraint violations (e.g., `DaysSupply <= 0` or negative paid amounts) are high severity because they directly break utilization metrics and downstream targeting features.
- Extreme utilization patterns require robust handling: flagging (and contextualizing by specialty/region) rather than naive removal.
- Missingness patterns concentrated in certain payer types or channels can reveal data collection or mapping gaps.

#### Visualizations to include (insert from notebook)

- *Figure 1:* Missingness by feature (bar chart)
- *Figure 2:* Missingness by subgroup (e.g., PlanType or Channel) (grouped bar chart / heatmap)
- *Figure 3:* Utilization scatter (e.g., `Qty` vs `DaysSupply` or `DaysSupply` vs `PaidAmount`) with flagged anomalies
- *Figure 4:* Before/after distribution plots (`DaysSupply`, `PaidAmount`, engineered `DailyDoseProxy`)

### Dataset 2: Housing (QC Core + Housing Plugin)

A second dataset demonstrates that the **QC Core is reusable**, while the housing plugin encodes domain semantics. The key outcome is that the same Detect–Fix–Communicate workflow transfers cleanly across domains while preserving domain-specific rules.

#### Expected findings

- Constraint violations (e.g., non-positive sale prices) are high severity because they directly break analysis/model assumptions.
- Extreme values (e.g., unusually large living area) require robust handling: flagging or capping with justification.
- Missingness patterns concentrated in certain neighborhoods can reveal data collection bias.

#### Visualizations to include (insert from notebook)

- Missingness by feature (bar chart)
- Missingness by Neighborhood (heatmap or grouped bar chart)
- `GrLivArea` vs `SalePrice` scatter with flagged outliers annotated
- Before/after distribution plots (`SalePrice`, `GrLivArea`)

Connecting back to theory (why results matter)

These results demonstrate the value of computational skepticism:

- Data quality issues are not cosmetic; they can change descriptive statistics, rankings, and model behavior.
  - Robust checks + auditable fixes produce more stable and credible insights.
  - Clear visualization improves trust and enables faster human judgment.
- 

## 7. References

Wickham, H. (2014). *Tidy Data*. Journal of Statistical Software, 59(10).

<https://www.jstatsoft.org/article/view/v059i10/772>

Breck, E., Polyzotis, N., Roy, S., Whang, S. E., & Zinkevich, M. (2019). *Data Validation for Machine Learning*. Proceedings of Machine Learning and Systems (MLSys).

[https://proceedings.mlsys.org/paper\\_files/paper/2019/hash/928f1160e52192e3e0017fb63ab65391-Abstract.html](https://proceedings.mlsys.org/paper_files/paper/2019/hash/928f1160e52192e3e0017fb63ab65391-Abstract.html)

Great Expectations. (n.d.). *Great Expectations Documentation (Expectation-based data validation)*.

<https://docs.greatexpectations.io/docs/home/>

Knafl, C. N. (2015). *Storytelling with Data: A Data Visualization Guide for Business Professionals*. Wiley.

<https://www.wiley.com/en-us/Storytelling%2Bwith%2BData%3A%2BA%2BData%2BVisualization%2BGuide%2Bfor%2BBusiness%2BProfessionals-p-9781119002253>