Photo by Fabrizio Conti on Unsplash

Starting v

This article introduces a concept for evaluating the dirtiness of a dataset, a topic that presents challenges due to the lack of a tangible score or loss function related to data cleaning. The primary objective here is to establish a metric that can effectively measure the cleanliness level of a dataset, translating this concept into a concrete optimisation problem.

Data cleaning is defined as a two-phase process:

1. First, **detecting data errors** such as formatting issues, duplicate records, and outliers;
2. Second, **fixing** these **errors**.

The evaluation of each phase typically relies on comparing a dirty dataset against a clean (ground truth) version, using classification metrics like recall, precision, and F1-score for error detection (see for example Can Foundation Models Wrangle Your Data?, Detecting Data Errors: Where are we and what needs to be done?) and accuracy or overlap-based metrics for data repair tasks (see Automatic Data Repair: Are We Ready to Deploy? or HoloClean: Holistic Data Repairs with Probabilistic Inference).
However, these metrics are task-specific and do not offer a unified measure for the overall cleanliness of a dataset that includes various types of errors.

This discussion is focused on **structured and tidy tabular datasets** (see Tidy Data | Journal of Statistical Software), **distinguishing data cleaning from broader data quality concerns** that include data governance, lineage, cataloguing, drift, and more.

The score blueprint

All the assumptions hereafter are the foundations the *Data Dirtiness Score* relies on. There are largely inspired by the article How to quantify Data Quality?. Of course, all of them could be debated and criticised but it is crucial to clearly state them to enhance discussions.

Data errors are tied to violated constraints, which arise from **expectations** about the data. For example, if the expectation is that the ID column should have no missing values, the presence of missing IDs would constitute a constraint violation.

No Expectation No Cry. **The absence of expectations means no impact on the score.** In other wotations, and thus, c

Data issues should be locateable to specific cells. The score relies on the ability to pinpoint errors to particular cells in the dataset.

Confidence scores for data errors. Not all data errors are equally certain. Each identified issue should have an associated confidence score, reflecting the probability or consensus around the error's validity, acknowledging that some issues might be subject to interpretation.

Uniform impact of cells on the overall score. Each cell in a dataset has an equal potential impact on the dirtiness score. Addressing an issue related to a given cell may resolve issues in others, suggesting a uniform distribution of cell weights in the score calculation.

A toy example for illustration

When examining a dataset, it's not uncommon to spot potential data quality issues at a glance. Consider the following simple dataset for analysis:

```
Student#,Last Name,First Name,Favorite Color,Age
1,Johnson,Mia,periwinkle,12
2,Lopez,Liam,blue,green,13
3,Lee,Isabella,,11
4,Fisher,Mason,gray,-1
5,Gupta,Olivia,9,102
6,,Robinson,,Sophia,,blue,,12
```

This example from the book Cleaning Data for Effective Data Science illustrates data quality issues within a dataset representing a 6th-grade class. This dataset includes multiple variables for each student, organised such that there are 6 students and 5 variables per student.

Upon inspection, certain entries might raise concerns due to apparent inconsistencies or errors:

- The entry for the student with `Student#` 2 (Lopez, Liam) appears to have an extra value in [boilerplate: You are signed out. Sign in with your member account (bh__@g__.com) to view other member-only stories. Sign in] ,green') have be value. Given the uncertainty, this issue is flagged with a 90% confidence level for further inspection.

- The next student, Isabella Lee, lacks a `Favorite Color` value. Given that this column should not have any missing entries, this issue is identified with 100% confidence for correction.

- The record for student number 4, Mason Fisher, lists an age of `-1`, an implausible value. This might represent a sentinel value indicating missing data, as it is common practice to use such placeholders. However, ages should be positive integers, necessitating a review of this entry.

- The row for student number 5, Olivia Gupta, while free from structural errors, presents an unusual case as several explanations are plausible. The `Favorite Color` and `First Name` fields might be swapped, considering `Olivia` can be both a name and a colour. Alternatively, the number `9` could represent a colour code, but this hypothesis lacks corroborating evidence. Moreover, an age of `102` for a 6th-grade student is highly improbable, suggesting potential typographical errors (e.g. `102` instead of `12`).

- The last row contains superfluous commas, indicating a possible data ingestion issue. However, aside from this formatting concern, the entry itself seems valid, leading to a high confidence level in identifying the nature of this error.

Following our guidelines to compute the dirtiness score, we can adopt a methodical approach by introducing a `DataIssue` class in Python, designed to encapsulate various aspects of a data issue:

```
@dataclass
class DataIssue:
    type_of_issue: str
    expectation: str
    constraint_violated: str
    confidence_score: float
    location: np.ndarray
```

To locate specific errors, a numpy array of size (6, 5) is utilised, where each element co ls, with 1 indicating

All the identified data issues are instantiated hereafter:

```
# Issue with Student# 2 - Extra value in 'Favorite Color'
issue_1 = DataIssue(
    type_of_issue="Extra Value",
    expectation="Single value in 'Favorite Color'",
    constraint_violated="It looks like two values ('blue,green') have been merg
    confidence_score=0.9,
    location=np.array([
                [0, 0, 0, 0, 0],
                [0, 0, 0, 1, 0],
                [0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0]
                ],
        )
)

# Issue with Student# 3 - Missing 'Favorite Color'
issue_2 = DataIssue(
    type_of_issue="Missing Value",
    expectation="No missing values in 'Favorite Color'",
    constraint_violated="Non-null constraint",
    confidence_score=1.0,
    location=np.array([
                [0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0],
                [0, 0, 0, 1, 0],
                [0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0]
                ],
        )
)

# Issue with Student# 4 - Implausible Age
issue_3 = DataIssue(
    type_of_issue="Implausible Value",
    expectation="Positive integer for 'Age'",
    constraint_violated="Positive integer constraint",
    confidence_score=1.0,
    location=np.array([
                [0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0],
```

```
                [0, 0, 0, 0, 1],
```



```
            ,
    )

    # Issues with Student# 5 - Multiple potential issues
    issue_4 = DataIssue(
        type_of_issue="Structural/Typographical Error",
        expectation="Consistent and plausible data entries",
        constraint_violated="The `Favorite Color` and `First Name` fields might be
        confidence_score=0.3,
        location=np.array([
                    [0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0],
                    [0, 0, 1, 1, 0],
                    [0, 0, 0, 0, 0]
                    ],
            )
    )

    issue_5 = DataIssue(
        type_of_issue="Typecasting error",
        expectation="`Favorite Color` must only contain values from known color str
        constraint_violated="`9` is not a valid colour name",
        confidence_score=0.9,
        location=np.array([
                    [0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0],
                    [0, 0, 0, 1, 0],
                    [0, 0, 0, 0, 0]
                    ],
            )
    )

    issue_6 = DataIssue(
        type_of_issue="Anomaly",
        expectation="Realistic age values for 6th-grade students",
        constraint_violated="An age of `102` is highly improbable",
        confidence_score=0.95,
        location=np.array([
                    [0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 1],
                    [0, 0, 0, 0, 0]
                    ],
```

```
        )
)
```



```
# Issue
issue_7 = DataIssue(
    type_of_issue="Formatting Error",
    expectation="Correct delimiter usage",
    constraint_violated="Duplicate commas as separators",
    confidence_score=1.0,
    location=np.array([
                [0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0],
                [1, 1, 1, 1, 1]
                ],
        )
)
```

The categorisation of multiple data errors into specific `DataIssue` instances can be somewhat subjective, similar to the nuances involved in bug reporting in software development. The fields— `type_of_issue`, `expectation`, and `constraint_violated` — serve to elucidate the nature of the error, facilitating understanding during investigations or reviews.
For computing the dirtiness score, the critical elements are the locations of the errors and the associated confidence scores. In this example, the confidence scores are estimated based on the perceived certainty of an error's presence.

> Repeated issues pointing to the same cells significantly increase the likelihood of a problem being present there.

Now that we have all the information we need, let's see how to calculate the dirtiness score for this small data set.

Calculation of the Data Dirtiness Score

> The *Data Dirtiness Score* represents the **expected fraction of cells** in a dataset **that contain errors.**

The theory and calculation for this score are elaborated in the `Score Theory` section of the appendix.

By using confidence scores for various issues as estimates for the independent probability principles to calculate *Dirtiness Score*.

Below is a Python function to calculate this metric based on a list of identified data issues:

```
def compute_data_dirtiness_score(data_issues: List[DataIssue]) -> float:
    """
    Computes the Data Dirtiness Score based on a list of data issues.
    Each issue's impact on data quality is represented by a confidence score
    and its location within the dataset.
    The function aggregates these impacts to estimate the overall 'dirtiness'
    of the dataset, with higher scores indicating lower quality.

    Parameters:
        data_issues: A list of DataIssue instances,
        each detailing a specific data quality issue.

    Returns:
        The overall Data Dirtiness Score for the dataset, as a float.
    """

    # Stack the probability arrays of a cell being error-free per issue
    stacked_error_free_probs = np.stack(
        [(1 - issue.confidence_score*issue.location) for issue in data_issues],
        axis=-1,
    )

    # Calculate the combined matrix probabilities of an issue for each cell
    probs_issue = 1 - np.prod(stacked_error_free_probs, axis=-1)

    # Find the average probability across all cells to get the dirtiness score
    data_dirtiness_score = np.mean(probs_issue)

    return data_dirtiness_score
```

Let’s compute the score for the data set presented earlier:

```
compute_data_dirtiness_score(data_issues)
```

> Data Dirtiness Score: 33.60%

To improve rs, such as correcting duplicate commas used as separators in the last row. Here is the new version of the dataset:

```
Student#,Last Name,First Name,Favorite Color,Age
1,Johnson,Mia,periwinkle,12
2,Lopez,Liam,blue,green,13
3,Lee,Isabella,,11
4,Fisher,Mason,gray,-1
5,Gupta,Olivia,9,102
6,Robinson,Sophia,blue,12
```

Let's recompute the score once again to see the improvement.

```
compute_data_dirtiness_score(data_issues)
```

> Data Dirtiness Score: 16.93%

Reevaluating the score post-correction reveals a significant improvement, halving the score due to the nature of the error affecting an entire row in a relatively small dataset.

In conclusion, this measure provides a quantitative means of monitoring and improving the cleanliness of our dataset by correcting iteratively identified data errors.

Next Steps and Challenges

Creating expectations or constraints for data can be challenging and costly due to the need for human labelling and domain knowledge. A solution is to automate the generation of constraints and data error detection, allowing humans to later review and adjust these automated constraints by either removing issues or modifying confidence scores. For that purpose, LLMs are really good candidates (cf. Jellyfish: A Large Language Model for Data Preprocessing, Can language models automate data wrangling? or Large Language Models as Data Preprocessors).

The likelihood of certain constraints and violations isn't always crystal-clear, which necessitate rts might not always n detecting these issues, having an estimated likelihood becomes particularly useful.

What about absent expectations or missed data errors? The effectiveness of error detection directly influences the cleanliness score and can lead to an overly optimistic value. However, there's a counterargument to consider: errors that are more difficult to detect, and thus more concealed, might not be as critical in their impact on data usability or downstream applications. This suggests that such errors should be assigned a lower confidence score when identified as issues, reflecting their reduced significance. While this approach may not be without its flaws, it serves to limit the influence of these overlooked errors on the overall dirtiness score by weighting their importance accordingly.

Another aspect to consider is the dynamic nature of the score. Addressing one issue could potentially affect other issues, raising questions about how to update the score efficiently without much hassle.

There's also the question of whether to include indexes and column names as part of the dataset cells when calculating the cleanliness score, as their accuracy can also affect the data cleaning process (see for example Column Type Annotation using ChatGPT).

Future articles in this series will explore various related topics, including a taxonomy of data errors, leveraging LLMs for automated issue detection, and strategies for data correction and repair. Stay tuned then!

References

- Can Foundation Models Wrangle Your Data?
- Detecting Data Errors: Where are we and what needs to be done?
- Automatic Data Repair: Are We Ready to Deploy?
- HoloClean: Holistic Data Repairs with Probabilistic Inference

- Tidy Data | Journal of Statistical Software
- How to
- Cleaning Data for Effective Data Science
- Jellyfish: A Large Language Model for Data Preprocessing
- Can language models automate data wrangling?
- Large Language Models as Data Preprocessors
- Column Type Annotation using ChatGPT

Score theory

Let's dive into the concept of calculating the *Data Dirtiness Score* for a dataset, denoted as $\mathcal{D}$. This dataset comprises I rows, representing individuals, and J columns, representing different variables.

We introduce a matrix X, which is of the same dimensions as $\mathcal{D}$, with I rows and J columns:

$$\begin{bmatrix} X_{11} & \cdots & X_{1J} \end{bmatrix}$$

$$X_{ij} \sim \mathcal{B}(\pi_{ij})$$

In this matrix, each element X_{ij} follows a Bernoulli distribution with parameter π_{ij}. The value of X_{ij} is set to 0 if the cell (i, j) in dataset $\mathcal{D}$ is free from data issues, and 1 if there is an issue, with the probability $\mathbb{E}[X_{ij}] = \pi_{ij}$ indicating the likelihood of an issue being present.

Next, we define a random variable Y that represents the proportion of cells in $\mathcal{D}$ that are problematic. The formula for Y is given by:

The *Data Dirtiness Score* is then the expected value of Y:

$$\text{Data Dirtiness Score} = \mathbb{E}[Y] = \frac{1}{I \times J} \sum_{i,j} \mathbb{E}[X_{ij}] = \frac{1}{I \times J} \sum_{i,j} \pi_{ij} = \overline{\pi}$$

To connect this back to our earlier discussion, the link between the confidence scores for each cell's data error and the probabilities π_{ij} is captured by the following relationship:

$$1 - \pi_{ij} = \prod_{i,j} (1 - \text{Confidence Score}_{ij})$$

This means that the probability of a cell being error-free is calculated as the product of the complements of the confidence scores for potential errors in that cell.

If all confidence scores are set to 1, indicating absolute certainty of errors, the dirtiness score simplifies to the proportion of cells with errors in the dataset. Calculating the dirtiness score or the cleanliness score for a dataset essentially yields the same insight, just from different perspectives. The formula for the *Data Cleanliness Score* is simply one minus the *Data Dirtiness Score*:

$$\text{Data Cleanliness Score} = 1 - \text{Data Dirtiness Score}$$

In this way, a dataset with no errors at all would have a cleanliness score of 100% and a dirtiness score of 0%.

Data Quality | Data Cleaning | Data Engineering | LIm | Data Science