DataCockpit: A Toolkit for Data Lake Navigation and Monitoring Utilizing Quality and Usage Information

Arpit Narechania
Georgia Institute of Technology
Atlanta, USA
arpitnarechania@gatech.edu

Surya Chakraborty
Georgia Institute of Technology
Atlanta, USA
schakraborty76@gatech.edu

Shivam Agarwal
Georgia Institute of Technology
Atlanta, USA
s.agarwal@gatech.edu

Atanu R Sinha
Adobe Research
Bengaluru, India
atr@adobe.com

Ryan A. Rossi
Adobe Research
San Jose, USA
ryrossi@adobe.com

Fan Du
Databricks
San Francisco, USA
dufan2013@gmail.com

Jane Hoffswell
Adobe Research
Seattle, USA
jhoffs@adobe.com

Shunan Guo
Adobe Research
San Jose, USA
sguo@adobe.com

Eunyee Koh
Adobe Research
San Jose, USA
eunyee@adobe.com

Alex Endert
Georgia Institute of Technology
Atlanta, USA
endert@gatech.edu

Shamkant Navathe
Georgia Institute of Technology
Atlanta, USA
sham@cc.gatech.edu

***Abstract*—Modern organizations amass their datasets into centralized repositories called data lakes, affording analytics as needed. The resultant scale and complexity of these data lakes, however, can make data navigation and monitoring challenging for users. We present DataCockpit, a Python toolkit that leverages datasets, usage logs, and associated meta-data to provision data usage and quality characteristics. DataCockpit computes these characteristics for each attribute (e.g., number of times it was queried for subsequent use in downstream applications) and record (e.g., number of non-missing, valid values) and aggregates them at the level of datasets. We develop a visual monitoring tool, powered by DataCockpit, and demonstrate how it can assist data / system administrators as well as end-users to effectively navigate and monitor a data lake. DataCockpit and the monitoring tool are available as open source software for developers to build custom monitoring applications on top of data lakes.**

***Index Terms*—data usage, data quality, monitoring, navigation, visualization, toolkit**

I. Introduction and Background

Modern organizations amass their data into centralized repositories called data lakes enabling users to perform analytics as needed [1]. Unfortunately, the inherent data overload due to this "load-first" philosophy poses several challenges in data navigation, discovery, and monitoring [2]–[5]. No single user knows about all the datasets, let alone what each one contains. This unfamiliarity leads to adverse consequences such as difficulties faced by users in discovering relevant attributes for data preparation, challenges faced by data administrators while monitoring the health and data governance policies of the data lake, and so on.

Prior work has utilized raw data [2], [6], meta-data [7], [8], and query-driven techniques [9]–[11] to support such data navigation and monitoring tasks. Existing proprietary software (e.g., Monte Carlo [12], Bigeye [13], Datafold [14]) as well as open-source tools (e.g. Elementary-Data [15], SQLLineage [16]) provide data profile, quality, and lineage information for data observability, monitoring, and pipeline optimization purposes. However, an important reason for the existence of data lakes is the *access* they provide to a large variety of datasets, an area where these tools fall short. These tools predominantly compute characteristics of the data, e.g., their *quality* [1], but not their *usage*, defined as the historical utilization characteristics of data across other users [17]. We address this 'gap' in this work.

Any single user can benefit by learning about the *usage* of different data (e.g., table attributes and records) across other users. Together with *quality*, usage information improves efficiency and effectiveness of downstream tasks. These benefits are evidenced in a rigorous user study of 36 users working with a visual data preparation and analysis tool, DataPilot [17]. DataPilot provisioned quality and usage information to users and was shown to help them select small, effective subsets from large, unfamiliar, tabular datasets.

In this paper, we present **DataCockpit**, a Python toolkit that utilizes quality and usage information to help users *navigate* and *monitor* data lakes. DataCockpit extends DataPilot's functionalities to a data lake comprising multiple relational datasets [18] and a logging framework [19]. While data lakes comprise heterogeneous (e.g., un-, semi-, structured) data, our

[1]Quality is defined as the validity and appropriateness of data required to perform certain analytics [17].

current focus is on one with relational databases with future plans to expand to other data sources.

DataCockpit computes these characteristics for each attribute (e.g., number of times the attribute was queried for subsequent use in downstream applications) and record (e.g., number of non-missing values, valid values) and assigns scores out of 100, that are then aggregated to a dataset-level. DataCockpit provides a customizable and extensible Python API to compute, persist, and query usage metrics such as who used which dataset, how, when, and why; and quality metrics namely *completeness*, *correctness*, *objectivity* [17], [20][2], *uniqueness* (number of distinct values), and *freshness* (timeliness of data for the task at hand) [20]. Unlike existing tools, DataCockpit provisions **both** usage and quality information across *multiple* granularities of data (i.e. attributes, records, databases).

Using DataCockpit, we developed a visual monitoring tool that presents usage and quality information with interactive affordances for data lake navigation and monitoring. We demonstrate its utility through multiple usage scenarios across different user groups: (a) data / system administrators to monitor the health of a data lake, (b) developers to build customized database applications, and (c) new or experienced end-users to flatten their learning curve or improve effectiveness while navigating a data lake, respectively. DataCockpit and the tool are released as open-source software at **https://github.com/datacockpit-org**.

The primary contributions of this work include:

1) **DataCockpit**, an open-source Python toolkit that provides **usage** and **quality** information from data lakes.
2) a data monitoring tool, powered by DataCockpit, to help users navigate and monitor data lakes.
3) use cases of this tool across multiple stakeholder types.

II. DataCockpit

We conducted a design study with fourteen experienced professionals (10 male, 4 female) from the software industry who performed data preparation and analysis for myriad tasks. We conducted semi-structured interviews to first understand their tasks and challenges and then led brainstorming and feedback sessions centered around the design and development of DataCockpit and the companion monitoring tool. Next, we describe how DataCockpit models these usage and quality information along with its architecture and API.

A. Modeling Data Usage Information

We posit that data lake *usage_logs* are similar to database management logs (e.g., MySQL's Query Log [21]), comprising *_id*, SQL *query*, *db* (database), *timestamp*, *user*, *app*lication.

_id	query	db	timestamp	user	app
1	SELECT prod.name, ...	org	1683743727	nroy	dash
2	INSERT INTO prod ...	org	1684901000	rbob	cli
3	UPDATE user SET role ...	org	1685329543	sguo	hr-app

[2]*completeness*: % of non-missing values, *correctness*: % of correct values, *objectivity*: the amount (in %) of lack of distortion in the data distribution.

We iterate through each *usage_log*, parse the SQL *query* [22], [23], and extract the *dataset*s (e.g., "product" table), *attribute*s (e.g., "name" column) and associated SQL *keyword*s (e.g., SELECT). We also extract filter conditions (specified via WHERE and HAVING clauses) to determine how frequently certain data records (rows) are accessed, identifying data 'hot-spots' (most-) and 'blind-spots' (least-used).

```
for log in usage_logs: # iterate over every usage log
  parsed_sql = SQLParser(log[“query”]) # parse the SQL query
  datasets = # from, join, updated (including those in subqueries)
  attributes = # select, update, where, group by, order by, having
  records = # exec query with(out) where, having, mark result set.
  user = log[“user”] # the user executing the query
  application = log[“app”] # the originating application
  timestamp = log[“timestamp”] # the query execution timestamp
```

This extracted information is then stored in new tables that end-users can query and get answers to questions such as: Which *attribute*s (or *dataset*s) are accessed most frequently? Are they SELECTed or UPDATEd, or both? Which user(s) access these and for which application(s)? When?

_id	db	data	attribute	keyword	timestamp	user	app
1	org	prod	name	SELECT	1683743727	nroy	dash
2	org	prod	-	INSERT	1684901000	rbob	cli
3	org	user	role	UPDATE	1685329543	sguo	hr-app

attribute	t_query_count	u_user_count	u_app_count	last_used	...
name	825	23	4	1683743727	...
role	2	1	1	1685329543	...

We precompute certain usage metrics for attributes (and records), e.g., *t(otal)_query_count*, *u(nique)_user_count*, *u_app_count*, *last_used*. If the data lake ingests data in batches, we can also aggregate usage across specific temporal *period*s (e.g., monthly *t_query_count*).

attribute	period	t_query_count	u_user_count	u_app_count	...
name	2023-05	43	5	5	...
role	2023-05	1	5	5	...

Scoring, Aggregation, and Customization. Each usage metric is first normalized to 0–100 for all attributes and records; this is its score. Then, the *overall* score is derived as the *max* of individual these scores. We do not use the *mean* because of variances in how the same data might be used across apps (e.g., regular versus one-off use) and low scores could potentially demotivate the user. These overall scores ($E_{1 \dots n}$ for 'n' attributes and records) are aggregated to higher granularities of data (e.g., dataset-level) using a *weighted average* (default weights = 1) as not all data are equally important, e.g, *t_query_count* is more important than *u_user_count*.

$$\textsc{AggrScore} = \frac{w_1 E_1 + w_2 E_2 + \ldots + w_n E_n}{w_1 + w_2 + \ldots + w_n} \qquad (1)$$

B. Modeling Data Quality Information

Complementing usage, we operationalize the data quality metrics – *completeness, correctness, objectivity* (from [17]) and include two additional metrics: *uniqueness* and *freshness* [20].

Completeness is the percentage of *non-missing values*, e.g., if 10 of 50 values are *nulls* or *empty strings*, then its completeness will be 100*(50-10)/50 = 80%. This metric can be computed for both attributes and records. We allow users to specify custom missingness criteria, e.g., ASCII-space (“ ”) or *NaN*. With completeness, users can detect sparse features (attributes) that may otherwise hinder an ML algorithm’s ability to make accurate predictions; and identify sparse records (rows) for examination, when needed.

Attribute Completeness

```
/* Compute percent of Non-Null values for all $N$ attributes
 * where $attr_i$ means the $i^{th}$ attribute for i ∈ 1...N. */
SELECT CAST(COUNT($attr_1$) AS FLOAT) ∗ 100 /
            CAST (COUNT(∗) AS FLOAT) AS ‘‘$attr_1$’’,
        CAST(COUNT($attr_2$) AS FLOAT) ∗ 100 /
            CAST (COUNT(∗) AS FLOAT) AS ‘‘$attr_2$’’,
        ...,
        CAST(COUNT($attr_N$) AS FLOAT) ∗ 100 /
            CAST (COUNT(∗) AS FLOAT) AS ‘‘$attr_N$’’
FROM dataset;
```

Record Completeness

```
/* Compute the percent of Non-Null values for all dataset records
 * where $attr_i$ means $i^{th}$ attribute where i ∈ 1...N. */
SELECT PrimaryKey, (CAST(
    (CASE WHEN $attr_1$ IS NULL THEN 0 ELSE 1 END) +
    (CASE WHEN $attr_2$ IS NULL THEN 0 ELSE 1 END) +
    ... +
    (CASE WHEN $attr_N$ IS NULL THEN 0 ELSE 1 END) AS
  FLOAT) ∗ 100 / $N$) AS ‘‘record_completeness’’
FROM dataset;
```

Correctness is the percentage of *correct* values based on pre-defined constraints and is computed for both attributes and records, for example, if 5 of 50 values for an attribute are incorrect, then its correctness is 100*(50-5)/50 = 90%. As correctness is subjective and context-dependent, we allow users to specify their own criteria and map them to SQL constraints using relations (>,<,=), range (BETWEEN), pattern matching (LIKE), and membership (IN) operators. Correctness allows users to assess accuracies of individual attributes, and act on incorrect information detected in records.

Attribute Correctness

```
/* Compute the percent of correct values for all $N$ attributes
 * where $attr_i$ means $i^{th}$ attribute for i ∈ 1...N. */
SELECT
  CAST(100 ∗ CAST(SUM(CASE WHEN Age BETWEEN 0 AND 150
    THEN 1 ELSE 0 END) AS FLOAT) / COUNT(Age) AS FLOAT),
  CAST(100 ∗ CAST(SUM(CASE WHEN Country IN (‘CA’,‘US’)
    THEN 1 ELSE 0 END) AS FLOAT) / COUNT(Country) AS FLOAT),
    ...,
  CAST(100 ∗ CAST(SUM(CASE WHEN $attr_N$ satisfies a condition
    THEN 1 ELSE 0 END) AS FLOAT) / COUNT($attr_N$) AS FLOAT)
FROM dataset;
```

Record Correctness

```
/* Compute the percent of correct values for all dataset records
 * where $attr_i$ means $i^{th}$ attribute where i ∈ 1...N. */
SELECT
  CAST(
    (CAST(100 ∗ CAST(
        SUM(CASE WHEN Age BETWEEN 0 AND 150
        THEN 1 ELSE 0 END) AS FLOAT) / COUNT(Age)
            AS FLOAT) +
    CAST(100 ∗ CAST(
        SUM(CASE WHEN Email LIKE ’%_@_%.__%’
        THEN 1 ELSE 0 END) AS FLOAT) / COUNT(Email)
            AS FLOAT) +
    ...)
  / $N$ AS FLOAT) AS ‘‘record_correctness’’
FROM dataset;
```

Objectivity is the extent to which values conform to a pre-specified or target distribution, e.g., if the *Gender* attribute has 105 males and 45 females, against expectations of a more equal gender distribution, then it is evidently skewed towards males and hence not objective. We approached this problem in two ways: (1) we use SQL to calculate percentage counts of *specific* attribute categories (e.g., *Gender=“Male”*) and then compare them against user-defined rules (e.g., percentage count of males should not be >60); this approach allows us to classify specific attributes as objective or not, assigning a dichotomous score of 0 or 100. (2) Alternatively, we calculate the respective counts across *all* attribute categories (via GROUP BY, COUNT) and then compute the Attribute Distribution (AD) metric as suggested by Wall et al. [24] (used in [25], [26]). AD quantifies the deviation between the observed and the target distributions along a scale from 0 (low, objective) to 1 (high deviation, unobjective). Objectivity is inapplicable at a record-level since each record corresponds to values across different attributes that are not comparable with each other.

Attribute Objectivity

```
/* Compute if values for attributes are objective (100=Yes, 0=No)
 * where $attr_i$ means $i^{th}$ attribute for i ∈ 1...N. */
SELECT CAST(100 ∗ (CASE WHEN
  (SELECT 100 ∗ CAST(COUNT(Gender) AS float) /
    (SELECT CAST(COUNT(Gender) AS float) FROM dataset)
  FROM dataset
  WHERE Gender = ‘‘Male’’) > 60 /* where 60 = some threshold */
    THEN 0 ELSE 1 END) AS FLOAT) AS ‘‘Is_Gender_Objective’’, ...,
/* compare other $attr_i$ distributions with a target or baseline */
FROM dataset;
```

Uniqueness is calculated as the number of distinct values for both attributes and records. We compute this metric using SQL (for attributes) as well as Python (for records). With uniqueness, users can determine an attribute’s entropy, i.e., the level of information contained in a value when considering all possible values; and assess the amount of duplicate values in each record (e.g., “contact_no” and “phone_no” have the same values), and assess schema effectiveness.

Attribute Uniqueness

```
/* Compute number of unique values for all $N$ attributes.
 * where $attr_i$ means $i^{th}$ attribute where i ∈ 1...N */
SELECT COUNT(DISTINCT $attr_1$) AS ‘‘$attr_1$’’,
        ...,
        COUNT(DISTINCT $attr_N$) AS ‘‘$attr_N$’’
FROM dataset;
```

Freshness is defined as the extent to which data are sufficiently up-to-date for the task at hand [20]. By default, we calculate it as the complement of the percentage difference (e.g., 50%) between the number of time units elapsed since the data record was ingested (e.g., fifteen days) and the end-user specified threshold for freshness (e.g., thirty days). Like objectivity, freshness scores can be dichotomous (100 or 0), based on the difference falling either within or outside the threshold. Note that DataCockpit currently supports freshness at a record-level only; attribute-level freshness requires tracking the schema (e.g., ALTER table operations), which is future work.

Record Freshness

```
/* Compute freshness of each record where ingestionDate is when
 * the record is inserted; freshnessThreshold is the unit of freshness
 * (e.g., 30 days); referenceDate is compared with ingestionDate. */
SELECT PrimaryKey,
       CASE WHEN DATEDIFF(referenceDate, ingestionDate)
        <= freshnessThreshold
            THEN 100 /* Within threshold */
            ELSE 0 /* Outside threshold */
            END
        AS ‘‘freshness’’
FROM dataset;
```

Scoring, Aggregation, and Customization. Like usage, each quality metric is also scored out of 100 for all attributes and records; because these are percentage-based by definition, they are already normalized. An *overall* quality score is also computed, as a *weighted average* of the individual scores (unlike the *overall* usage score that uses *maximum*); the weights default to 1 but can be customized, e.g., for "timestamp", *uniqueness* may be assigned a smaller weight than *freshness* because timestamps will be unique; them being recent or not is perhaps more important. Lastly, all attribute- and record- level scores are aggregated to higher granularities of data (e.g., dataset-level), also using customizable weighted averages.

C. Python Toolkit Architecture and API

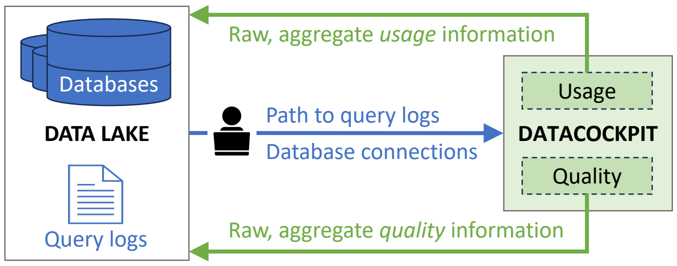


Fig. 1: DataCockpit architecture diagram illustrating how it can be integrated into existing data lakes.

DataCockpit is an open-source Python toolkit to enable developers to compute, persist, and later access quality and usage information, and build custom applications for their own data lakes. Given the path to *usage_logs* and appropriate database connections as input, DataCockpit computes raw and aggregate quality and usage information, and persists it back into the source data lake (Figure 1). The Python code below shows how to interact with the toolkit.

```
from datacockpit import DataCockpit
# Setup database connection with SQL engine and log-file CSV
dcp_obj = DataCockpit(engines=[array_of_sqlalchemy_engines],
        logs_paths=[array/of/your/logs/path.csv])
# Compute and persist quality & usage metrics
dcp_obj.compute_usage(levels=None, metrics=None)
dcp_obj.compute_quality(levels=None, metrics=None)
# Retrieve computed information for use in downstream apps
usage_info = dcp_obj.get_usage(**kwargs)
quality_info = dcp_obj.get_quality(**kwargs)
```

First, to connect to the data lake, we create SQLAlchemy[3] engines, one for each relational database. Second, we create an object, dcp_obj, of the DataCockpit class, by passing arrays of these engines and their corresponding logs_path as arguments. In this example, we both read from and write to the same database, however in practice, a separate database connection can be provided for writing the meta-data. logs_path points to the location where the historical usage logs (SQL queries and meta-data such as the querying user and the timestamp) are saved in a CSV file. The next two statements call methods to compute_usage() and compute_quality() with parameters to support different levels (e.g., subset of ["attribute", "record", "dataset", "database"]) and metrics (e.g., subset of ["correctness", ..., "freshness"]). These compute_*() commands also persist the computed metrics to a new database for retrieval. The get_usage(**kwargs) and get_quality(**kwargs) functions fetch these persisted metrics for building custom applications, where ***kwargs* are optional keyword arguments to seek only a subset of the computed metrics at specific aggregation levels.

[3]SQLAlchemy [27] is a popular Object-Relation Mapping (ORM) system that abstracts the underlying database engine allowing developers to write database-agnostic code.

III. Visual Monitoring Tool

A. Implementing the User Interface

Figure 2 shows the user interface and it consists of two main tabs.

Data Lake View. This view is the landing page and provides an overview of a single data lake (e.g., "Asia/Pacific") configurable via the dropdown. It includes an interactive table with information on constituent datasets, e.g., {"Id", "Name"}. The last two columns correspond to *overall* "Quality" and "Usage" scores, heuristically classified as *high* ● ($\geq$90), *medium* ● ($\geq$67 but <90), and *low* ●, using the same cutoffs as DataPilot [17] [4]. Users can utilize this information along with search, filter, and pagination interaction affordances to (1) **navigate**: strategically explore the datasets in the data lake based on their quality and/or usage (not by their name and/or create/update timestamps), (2) **discover**: find high quality, high usage, relevant datasets for a downstream application, (3) **monitor** the 'health' of the data lake, and (4) **housekeep**: find low quality, low usage irrelevant datasets for archival. Clicking a table row presents additional details about the corresponding dataset in the *Dataset View*.

Dataset View. The **Overview and Preview (a)** views present factual information for the selected dataset, as in the *Data Lake View*, along with five sample records (as a dataset preview).

The **Quality and Usage (b)** views visualize the computed quality and usage scores as colored glyphs: *high* ● ($\geq$90), *medium* ● ($\geq$67 but <90), and *low* ●. For example, Figure 2 shows that the "Duma" dataset has a *correctness* of 100% ● and has been used more to generate reports (70) ● and less to build dashboards (65) ●.

The **Evolution over Time (c)** view visualizes the evolution of overall usage and quality scores for this dataset over time. These scores are (re)computed whenever new records (a new batch of data) are appended to the dataset (even when new datasets are ingested), and/or on a scheduled (e.g., weekly) basis, helping users monitor the health of datasets.

The **Attribute and Record Explorer (d)** view presents quality and usage information for each attribute in an interactive list or tree visualization (Figure 2), and record as a tabular visualization. The tree visualization is useful for hierarchical data schemas (e.g., "placeContext.geo.city", "placeContext.geo.point.latitude") and lets users pan, zoom, expand, and collapse attribute nodes to promote overview first and details on demand visual exploration [28]. Nodes can be colored based on the quality or usage scores of the corresponding attributes. Node label colors correspond to whether an attribute in the mapped schema is in the dataset (black) or not (gray). Hovering an attribute node (e.g., "city" in Figure 2) shows corresponding quality and usage scores in a tooltip. For record-level information, a datatable provides similar capabilities.

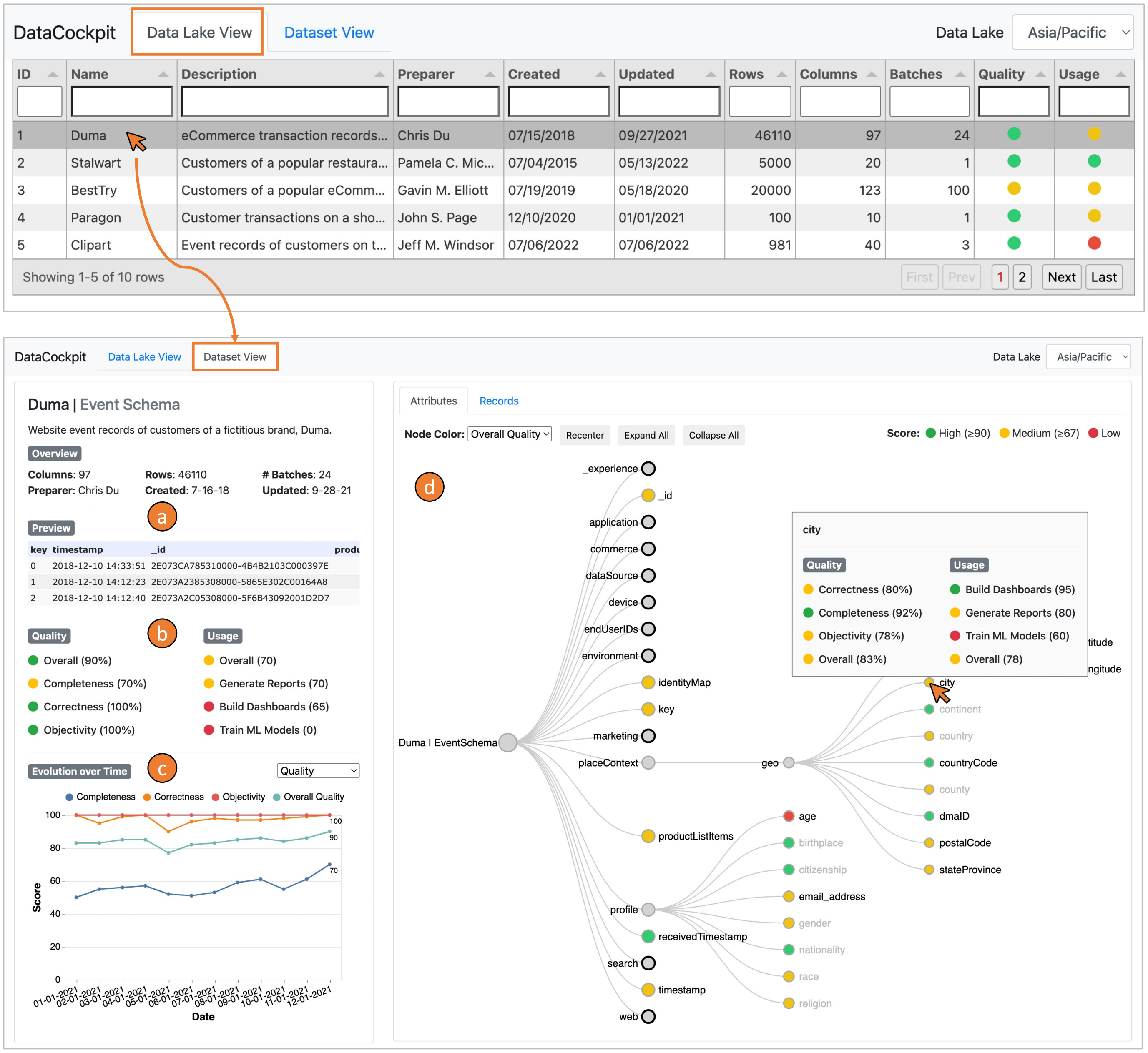
Implementation. We developed this interface using a client-server architecture where the client-side is written using HTML and JavaScript; and the server-side uses Python Flask [29]. The client-side collects and passes parameters (e.g., ***kwargs*) to the server-side over HTTP REST. The resultant JSON outputs from server-side are passed back and parsed in JavaScript to render the user interface: text and colored glyphs of usage and quality scores, temporal multi-series line chart, and the interactive tree visualization using D3.js [30].

B. Use Cases Envisioned

Unlike existing tools, DataCockpit provides monitoring capabilities on top of navigation capabilities, for multiple datasets, to support data / system administrators and developers. Like existing tools, it also continues to serve end-users.

Data / System Administrators. Administrators can efficiently find low quality, low recent usage datasets and potentially mark them for archival (cold storage); on the other hand, administrators can also

[4]Shown quality and usage information is simulated for this demonstration.


DataCockpit | Data Lake View | Dataset View | Data Lake: Asia/Pacific

ID	Name	Description	Preparer	Created	Updated	Rows	Columns	Batches	Quality	Usage
1	Duma	eCommerce transaction records...	Chris Du	07/15/2018	09/27/2021	46110	97	24	🟢	🟡
2	Stalwart	Customers of a popular restaura...	Pamela C. Mic...	07/04/2015	05/13/2022	5000	20	1	🟢	🟢
3	BestTry	Customers of a popular eComm...	Gavin M. Elliott	07/19/2019	05/18/2020	20000	123	100	🟡	🟡
4	Paragon	Customer transactions on a sho...	John S. Page	12/10/2020	01/01/2021	100	10	1	🟢	🟡
5	Clipart	Event records of customers on t...	Jeff M. Windsor	07/06/2022	07/06/2022	981	40	3	🟢	🔴

Showing 1-5 of 10 rows

DataCockpit | Data Lake View | Dataset View | Data Lake: Asia/Pacific

Duma | Event Schema

Website event records of customers of a fictitious brand, Duma.

Overview

Columns: 97 **Rows**: 46110 **# Batches**: 24
Preparer: Chris Du **Created**: 7-16-18 **Updated**: 9-28-21

Preview

key	timestamp	_id	produ
0	2018-12-10 14:33:51	2E073CA785310000-4B4B2103C000397E	
1	2018-12-10 14:12:23	2E073A2385308000-5865E302C00164A8	
2	2018-12-10 14:12:40	2E073A2C05308000-5F6B43092001D2D7	

Quality	Usage
🟢 Overall (90%)	🟡 Overall (70)
🟡 Completeness (70%)	🟡 Generate Reports (70)
🟢 Correctness (100%)	🔴 Build Dashboards (65)
🟢 Objectivity (100%)	🔴 Train ML Models (0)

Attributes | Records

Node Color: Overall Quality | Recenter | Expand All | Collapse All

Score: 🟢 High (≥90) 🟡 Medium (≥67) 🔴 Low

city

Quality	Usage
🟡 Correctness (80%)	🟢 Build Dashboards (95)
🟢 Completeness (92%)	🟡 Generate Reports (80)
🟡 Objectivity (78%)	🔴 Train ML Models (60)
🟡 Overall (83%)	🟡 Overall (78)

Fig. 2: Visual Monitoring Tool: the **Data Lake View** lists all datasets in the data lake; the **Dataset View** provides additional information (e.g., a preview) about a specific dataset (a), corresponding quality and usage scores (b), temporal evolution of these scores (c), and an interactive visualization showing attribute and record-level quality and usage information (d).

improve retrieval times for frequently used datasets (e.g., by setting up custom indexes). Executing DataCockpit separately on an existing dataset and an incoming batch can reveal changes in quality scores, and when the change is sufficiently negative, can help users to action and stop 'bad data' from ingestion into the system.

Developers. DataCockpit's open-source nature allows developers to create usage and quality powered tracking and monitoring capabilities into their own new tools, enabling end-users to better navigate the corresponding data environments.

End-users. Even experienced end-users who perform data preparation and analysis tasks may not be aware of usage and quality metrics of *all* datasets in their data lake. DataCockpit affords awareness, thereby improving effectiveness of their tasks. They can re-use 'tried, tested, and effective' attributes from past analysis and dashboards for future ones. By computing quality and usage scores for each incoming data batch and cumulatively for the entire dataset, these users can highlight shifting trends in the data (e.g., "age" is being queried a lot). New users can efficiently learn these metrics across all datasets, flattening their learning curve.

IV. Discussion, Limitations, and Future Work

DataCockpit currently supports limited data usage and quality metrics; we plan to implement more metrics (e.g., *consistency*, *accuracy* [20], [31]–[36]) and also support advanced computational abilities (e.g., anomaly detection). Next, DataCockpit currently supports diverse end-user needs by allowing developers to configure various parameters (e.g., missingness criteria for *completeness*) and thresholds (e.g., expiry criteria for *freshness*); as this manual configuration can sometimes be tedious, future work may provide out of the box defaults or elicit them from end-users via the user interface of the monitoring tool. We also plan to make DataCockpit privacy friendly by serving user requests based on an organization's data governance policy. Lastly, DataCockpit currently supports SQLAlchemy-based database connections and CSV-based usage logs; we will support other semi-, un-, and structured data sources (e.g., JSON, Parquet), and external tool integrations (e.g., Snowflake [37]) in the future. Our goal remains to offer a customizable, open-source alternative to existing solutions (e.g., Collibra [38]).

V. Conclusion

We leveraged datasets, usage logs, and associated meta-data within a data lake of relational databases to compute usage and quality information across attributes, records, and databases. We implemented these in a Python toolkit, **DataCockpit**, and built a visual monitoring tool to demonstrate its use during data lake navigation and monitoring for data / system administrators, developers, and end-users.

Acknowledgments

We thank our interviewees, members of the Georgia Tech Database Group and Georgia Tech Visualization Lab, and anonymous reviewers for providing helpful feedback at different stages of this work.

References

[1] L. Gitelman, *"Raw Data" Is An Oxymoron*. MIT press, 2013.

[2] D. Deng, R. C. Fernandez, Z. Abedjan, S. Wang, M. Stonebraker, A. K. Elmagarmid, I. F. Ilyas, S. Madden, M. Ouzzani, and N. Tang, "The Data Civilizer System," in *CIDR*, 2017. [Online]. Available: https://dblp.org/rec/conf/cidr/DengFAWSEIMO017.html

[3] R. Castro Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker, "Aurum: A Data Discovery System," in *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, 2018, pp. 1001–1012. [Online]. Available: https://doi.org/10.1109/ICDE.2018.00094

[4] F. Nargesian, K. Q. Pu, E. Zhu, B. Ghadiri Bashardoost, and R. J. Miller, "Organizing Data Lakes for Navigation," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: Association for Computing Machinery, 2020, p. 1939–1950. [Online]. Available: https://doi.org/10.1145/3318464.3380605

[5] F. Nargesian, E. Zhu, R. J. Miller, K. Q. Pu, and P. C. Arocena, "Data Lake Management: Challenges and Opportunities," *Proceedings of the VLDB Endowment*, vol. 12, no. 12, p. 1986–1989, aug 2019. [Online]. Available: https://doi.org/10.14778/3352063.3352116

[6] R. J. Miller, F. Nargesian, E. Zhu, C. Christodoulakis, K. Q. Pu, and P. Andritsos, "Making Open Data Transparent: Data Discovery on Open Data," *IEEE Data Engineering Bulletin*, vol. 41, no. 2, pp. 59–70, 2018. [Online]. Available: http://sites.computer.org/debull/A18june/p59.pdf

[7] A. Halevy, F. Korn, N. F. Noy, C. Olston, N. Polyzotis, S. Roy, and S. E. Whang, "Goods: Organizing Google's Datasets," in *Proceedings of the 2016 International Conference on Management of Data*. New York, NY, USA: Association for Computing Machinery, 2016, p. 795–806. [Online]. Available: https://doi.org/10.1145/2882903.2903730

[8] "Monosi," 2023. [Online]. Available: https://monosi.dev/

[9] W. Brackenbury, R. Liu, M. Mondal, A. J. Elmore, B. Ur, K. Chard, and M. J. Franklin, "Draining the Data Swamp: A Similarity-Based Approach," in *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3209900.3209911

[10] M. J. Cafarella, A. Halevy, and N. Khoussainova, "Data Integration for the Relational Web," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, p. 1090–1101, 2009. [Online]. Available: https://doi.org/10.14778/1687627.1687750

[11] Y. Zhang and Z. G. Ives, "Finding Related Tables in Data Lakes for Interactive Data Science," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: Association for Computing Machinery, 2020, p. 1951–1966. [Online]. Available: https://doi.org/10.1145/3318464.3389726

[12] "Monte Carlo Data," 2023. [Online]. Available: https://www.montecarlodata.com/

[13] "Bigeye," 2023. [Online]. Available: https://www.bigeye.com/

[14] "Datafold," 2023. [Online]. Available: https://www.datafold.com/

[15] "Elementary Data," 2023. [Online]. Available: https://www.elementary-data.com/

[16] "SQLLineage," 2023. [Online]. Available: https://github.com/reata/sqllineage

[17] A. Narechania, F. Du, A. R. Sinha, R. Rossi, J. Hoffswell, S. Guo, E. Koh, S. B. Navathe, and A. Endert, "DataPilot: Utilizing Quality and Usage Information for Subset Selection during Visual Data Preparation," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3544548.3581509

[18] R. Elmasri, S. B. Navathe, R. Elmasri, and S. Navathe, *Fundamentals of Database Systems*. Springer, 2000.

[19] S. Dumais, R. Jeffries, D. M. Russell, D. Tang, and J. Teevan, "Understanding User Behavior Through Log Data and Analysis," in *Ways of Knowing in HCI*. Springer, 2014, pp. 349–372.

[20] L. L. Pipino, Y. W. Lee, and R. Y. Wang, "Data Quality Assessment," *Communications of the ACM*, vol. 45, no. 4, p. 211–218, 2002. [Online]. Available: https://doi.org/10.1145/505248.506010

[21] "The General Query Log," 2023. [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/query-log.html

[22] "sqlparse," 2023. [Online]. Available: https://github.com/andialbrecht/sqlparse

[23] "sql-metadata," 2023. [Online]. Available: https://pypi.org/project/sql-metadata/

[24] E. Wall, L. M. Blaha, L. Franklin, and A. Endert, "Warning, Bias May Occur: A Proposed Approach to Detecting Cognitive Bias in Interactive Visual Analytics," in *2017 IEEE Conference on Visual Analytics Science and Technology (VAST)*, 2017, pp. 104–115. [Online]. Available: https://doi.org/10.1109/VAST.2017.8585669

[25] A. Narechania, A. Coscia, E. Wall, and A. Endert, "Lumos: Increasing Awareness of Analytic Behavior during Visual Data Analysis," *IEEE Transactions on Visualization and Computer Graphics*, vol. 28, no. 1, pp. 1009–1018, 2022. [Online]. Available: https://doi.org/10.1109/TVCG.2021.3114827

[26] E. Wall, A. Narechania, A. Coscia, J. Paden, and A. Endert, "Left, Right, and Gender: Exploring Interaction Traces to Mitigate Human Biases," *IEEE Transactions on Visualization and Computer Graphics*, vol. 28, no. 1, pp. 966–975, 2022. [Online]. Available: https://doi.org/10.1109/TVCG.2021.3114862

[27] "SQLAlchemy," 2022. [Online]. Available: https://www.sqlalchemy.org/

[28] B. Shneiderman, "The Eyes Have It: A Task By Data Type Taxonomy for Information Visualizations," in *Proceedings 1996 IEEE Symposium on Visual Languages*, 1996, pp. 336–343. [Online]. Available: https://doi.org/10.1109/VL.1996.545307

[29] Pallets, "Flask," https://palletsprojects.com/p/flask/, 2010.

[30] M. Bostock, V. Ogievetsky, and J. Heer, "D^3: Data-Driven Documents," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2301–2309, 2011. [Online]. Available: https://doi.org/10.1109/TVCG.2011.185

[31] "Data Quality — Part 2: Vocabulary," 2022. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso:8000:-2:ed-5:v1:en

[32] "ISO/IEC 25012," 2008. [Online]. Available: https://iso25000.com/index.php/en/iso-25000-standards/iso-25012

[33] "ISO 19157-1:2023; Geographic information — Data quality — Part 1: General requirements," 2023. [Online]. Available: https://www.iso.org/standard/78900.html

[34] "ISO 19115-1:2014; Geographic information — Metadata — Part 1: Fundamentals," 2014. [Online]. Available: https://www.iso.org/standard/53798.html

[35] F. Sidi, P. H. Shariat Panahy, L. S. Affendey, M. A. Jabar, H. Ibrahim, and A. Mustapha, "Data quality: A Survey of Data Quality Dimensions," in *2012 International Conference on Information Retrieval & Knowledge Management*, 2012, pp. 300–304. [Online]. Available: https://doi.org/10.1109/InfRKM.2012.6204995

[36] N. Laranjeiro, S. N. Soydemir, and J. Bernardino, "A Survey on Data Quality: Classifying Poor Data," in *2015 IEEE 21st Pacific Rim International Symposium on Dependable Computing (PRDC)*, 2015, pp. 179–188. [Online]. Available: https://doi.org/10.1109/PRDC.2015.41

[37] "Snowflake," 2023. [Online]. Available: https://www.snowflake.com/en/

[38] "Collibra," 2023. [Online]. Available: https://www.collibra.com/