Study and analysis of the performance of JValue Open Data Service as part of a data pipeline supporting an online learning model

MASTER'S THESIS

Shady Hegazy

Submitted on 2 August 2022

Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik
Professur für Open-Source-Software

Supervisor:
Prof. Dr. Dirk Riehle, M.B.A.

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Versicherung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 2 August 2022

License

Erlangen, 2 August 2022

Abstract

Open data has been known for having *data quality* issues that require complex *data cleansing* and *data transformation* in order to be usable for *data analysis*, *data visualization*, training *machine learning* algorithms, and other *data science* activities. Open Data Service (ODS) is a software project that aims at creating an interface for reliable and safe consumption of *open data*. It does so by providing the necessary tooling and infrastructure needed for collaboration on eliminating *open data* usability obstacles. ODS underwent several cycles of development to better serve its purposes, which include functioning as an *extract, transform, load* (ETL) tool to consume *open data* from different sources and adapt it to different needs. In this work we evaluate and analyse ODS performance in that regard. Specifically, as part of a data pipeline supporting a real-world *data science* application.

Contents

List of Figures

List of Tables

Acronyms

ODS Open Data Service

ETL *extract, transform, load*

OGD Open Government Data

KDD Knowledge Discovery in Databases

CRISP-DM CRoss-Industry Standard Process for Data Mining

SEMMA Sample, Explore, Modify, Model, Assess

GUI graphical user interface

SIG Special Interest Group

RAMSYS Rapid Collaborative Data Mining System

TDSP Microsoft Team Data Science Process

TDWI The Data Warehousing Institute

ROLAP Relational Online Analytical Processing

MOLAP Multidimensional Online Analytical Processing

BI Business Intelligence

FCM Firebase Cloud Messaging

API Application Programming Interface

OOP Object-Oriented Programming

UML Unified Modeling Language

1 Introduction

The development of ODS as a streamlining interface between *open data* providers and consumers has already resulted in subsequent working versions that can be deployed and relied upon in a production setting. This progress opened up the question about the extent to which ODS can actually fit in data-dependent projects and applications, and the spectrum of data consumption activities it can support. In addition, as ODS is getting increasingly adopted and incorporated in applications, its development cycles require more adaptations and enhancements to better serve the widening scope of data consumption activities it supports. *Data science* applications are among the most important and data-intensive applications. Hence, a need for an evaluation of ODS as part of a *data science* pipeline has become more stressing for ODS development and adoption.

This thesis work focuses on creating a *data science* application that can utilize a wide spectrum of ODS capabilities as an ETL, while exposing the capabilities it still lacks and the areas in need of improvement. The methodology followed throughout this work comprises the following steps:

- Study the *data science* process to identify the main activities entailed in a *data science* project, which ODS will be required to support.
- Research evaluation criteria and desirable qualities of ETL tools with an inclination to focus on *data science* relevant features.
- Model a highly-performant ETL for *data science*.
- Model current capabilities of ODS v2.
- Curate and engineer requirements for an evaluation application based on head-to-head matching of elicited ETL quality metrics, the model for a highly-performant ETL for *data science*, and ODS capabilities model.
- Generate viable project scenarios of candidate evaluation applications.
- Select one project scenario through a comparison of viable candidates based on feasibility of requirements fulfillment.

- Design and implement the evaluation application.
- Evaluate ODS performance against predefined criteria.
- Conclude recommendations for improving ODS performance as an ETL for *data science*.

This work is spread across 6 chapters, including:

- *Chapter 2 (Fundamentals)* expands on the foundational concepts that will be mentioned and discussed throughout the following chapters.
- *Chapter 3 (Requirements Engineering)* explains steps taken in order to accomplish each phase of the requirements engineering process. It also discusses the selection of a project scenario for the evaluation application.
- *Chapter 4 (Architecture, Design, and Implementation)* lays out the architectural, design, and implementation specifications of the evaluation application.
- *Chapter 5 (Results)* discusses the observations made through the evaluation process and lists the concluded recommendations in an actionable presentation style.
- *Chapter 6 (Conclusion)* provides a final summary and conclusion statement for this work.

2 Fundamentals

In this chapter we clarify some of the main concepts and entities that are mentioned and discussed throughout this work. We follow an inside-out approach, expanding on the core concepts first.

2.1 *Open data*

2.1.1 In search for *open data* definition

The term *open data* may seem self-explanatory at the first glance. However, there has been no unified definition of it in academic literature so far (Piovesan, 2015). The term *open data* tends to be confused with the term Open Government Data (OGD). This is a result of the fact that governments public data publishing policies had major influence on the development, adoption, and fostering of the concept of *open data* (Hickmann Klein et al., 2017; Kvamsdal, 2017).

> For example, many historical accounts of *open data* in press pieces, blogs, government sources and popular literature centre around two incidents: a meeting of "open government advocates" to draw up a definition of open government data in Sebastopol, California in December 2007, followed by President Barack Obama's announcement in support of open government just over a year later on his first day in office in January 2009 and the subsequent launch of the US's government data portal Data.gov (Gray, 2014, p. 4).

Searching the literature for a definition, we noticed that there exists a mosaic of definitions for *open data*, with each definition addressing only some parts of the big picture.

2.1.2 Understanding *open data*

As a single comprehensive definition of *open data* could not be reached, we may infer a clear understanding of the concept from the metrics that has been put together to scale the degree of *openness* of data. Berners-Lee (2006) proposed an intuitive *5-stars* openness scheme that considers five metrics to evaluate the openness of a data initiative. This evaluation scheme has been practically considered the *de facto* standard for measuring data openness, despite being focused on the format and the encoding of the published data (Vetro et al., 2016). Table 2.1 lists the 5 principles and the corresponding stars as worded in (Berners-Lee, 2006). Under this scheme, the minimum requirement for a data to be open is to be published under an open license, which is the first and most important star. The successive 4 stars are awarded against fulfilling features that make the data more publicly usable.

Table 2.1: *5-stars* data openness evaluation scheme

Stars	Action
⋆	Available on the web (whatever format) but with an open licence, to be *open data*
⋆ ⋆	Available as machine-readable structured data (e.g. excel instead of image scan of a table)
⋆ ⋆ ⋆	as (2) plus non-proprietary format (e.g. CSV instead of excel)
⋆ ⋆ ⋆ ⋆	All the above plus, Use open standards from W3C (RDF and SPARQL) to identify things, so that people can point at your stuff
⋆ ⋆ ⋆ ⋆ ⋆	All the above, plus: Link your data to other people's data to provide context

Since Berners-Lee (2006) has introduced the aforementioned *5-stars* scheme, many definitions and evaluation schemes have been put forward. These newer schemes had broader perception of data openness metrics, which can expand our understanding of the concept. Piovesan (2015) described 15 characteristics that define "optimal open government data". Table 2.2 lists those characteristics as in (Piovesan, 2015). Compared to the *5-stars* scheme, the *15-characteristics* scheme requires more depth and breadth in the assessment of data openness. It incorporates the presence of context information (metadata) of the data in evaluation as an openness metric, which is critical for data re-usability. It also incorporates important quality metrics such as risk-free data consumption, version recentness of the data publication, accuracy, and fitness for publicity. These metrics accommodate more needs and requirements of *open data* consumers under wider spectrum of usage scenarios, compared to the brief *5-stars* approach which revolved around format and license.

In an effort to create a general *open data* quality assurance approach out of the available standards and schemes that had different scopes with varying width and focus, research efforts resulted in the subsequent publishing of generalized

Table 2.2: Optimal *open government data* (Piovesan, 2015).

1. Data should exist..	6. ..in a machine readable (open) format..	11. ..under open license..
2. ..in digital form..	7. ..available in bulk..	12. ..up to date..
3. ..publicized..	8. ..complete of context information..	13. ..risk-free..
4. ..online..	9. ..URIs..	14. ..with no meaning conflict..
5. ..for free..	10. ..and linked to other data [LOD]..	15. ..and allow for user feedback.

standards and frameworks for ensuring *open data* quality. Wilkinson et al. (2016) have put together a set of guidelines for *open data* quality assurance, these were called the *FAIR* principles. *FAIR* principles represented a consolidation of the previous sets of principles that have been developed separately with different focuses. Although initially focused on scholarly data, *FAIR* principles were generalized to address most data publishing activities. This generalization resulted in the wide adoption that *FAIR* principles gained afterwards (Mons et al., 2020). One important aspect of the *FAIR* rules is that it is technology- and architecture-independent: "These high-level FAIR Guiding Principles precede implementation choices, and do not suggest any specific technology, standard, or implementation-solution" (Wilkinson et al., 2016). The *FAIR* principles can be summarized, using adaptations from (Wilkinson et al., 2016), as follows:

- To be Findable:
  - F1. (meta)data are assigned a globally unique and persistent identifier
  - F2. data are described with rich metadata (defined by R1 below)
  - F3. metadata clearly and explicitly include the identifier of the data it describes
  - F4. (meta)data are registered or indexed in a searchable resource
- To be Accessible:
  - A1. (meta)data are retrievable by their identifier using a standardized communications protocol
    * A1.1 the protocol is open, free, and universally implementable
    * A1.2 the protocol allows for an authentication and authorization procedure, where necessary

    - A2. metadata are accessible, even when the data are no longer available

- To be Interoperable:
    - I1. (meta)data use a formal, accessible, shared, and broadly applicable language for knowledge representation
    - I2. (meta)data use vocabularies that follow FAIR principles
    - I3. (meta)data include qualified references to other (meta)data
- To be Reusable:
    - R1. meta(data) are richly described with a plurality of accurate and relevant attributes
        * R1.1. (meta)data are released with a clear and accessible data usage license
        * R1.2. (meta)data are associated with detailed provenance
        * R1.3. (meta)data meet domain-relevant community standards

We may now have formed a clearer understanding of the concept of *open data*. Inferring from the aforementioned openness standards, we can conclude that data is as open as it is equipped for maximum reusability. Thus designing solutions and initiatives that deal with *open data* should focus on increasing data reusability as a major design evaluation criteria, as it is the central goal of any *open data* initiative.

2.2 Extract-Transform-Load (ETL)

Data applications consume and use large amounts of data that come from different sources and in different formats. In order to ensure high availability and performance of these applications, it needs to be supported by data processing systems that can reliably consolidate data from different sources into the desired destination. ETL processes carry out that role as they can be "used to migrate heterogeneous data from one or more data sources into a target system to form data repositories, data marts, or data warehouses" (Albrecht & Naumann, 2009, p. 1). "ETL was born on the first day that a programmer constructed a program that takes records from a certain persistent file and populates or enriches another file with this information" (Vassiliadis & Simitsis, 2009, p. 2). In fact, ETL processes include so much more than the name indicates.

> As an acronym, however, ETL only tells part of the story. ETL tools also commonly move or transport data between sources and

> targets, document how data elements change as they move between source and target (i.e., meta data), exchange this meta data with other applications as needed, and administer all run-time processes and operations (e.g., scheduling, error management, audit logs, and statistics). A more accurate acronym might be EMTDLEA! (Wayne Eckerson & Colin White, 2003, p. 7).

ETL processes, in general, are processes that a target data undergoes in order to be ready for consumption for the desired application. It consists of 3 phases:

- Extraction
- Transformation
- Loading

In the first phase, data is fetched from different sources into the staging area. As the data come from different sources, it comes with a big deal of technical heterogeneity that needs to be detangled before pushing the data to the staging area and the next steps. The extraction phase ideally also works on increasing the relevancy of the imported data.

In the second phase, the staged data undergo the most important and necessary transformations to be completely ready for consumption. During this phase, the imported data undergo the most crucial and vital processing. Special components rectify the syntactical and semantical heterogeneities so that data con be mapped into a unified schema and model. Imported data then undergoes cleansing processes to mediate any cavities and act on errors in order to bring it to a common standard. A wide variety of components and processes can be part of the transformation phase such as: duplicate data consolidation, data aggregation or combination, data validation, and so forth (Albrecht & Naumann, 2009).

In the third phase, cleaned, preprocessed, and consolidated data is loaded into the target system or destination. The data can be loaded directly into the data application for consumption, but it is commonly the case that data is loaded by the ETL system into a data warehouse or a specialized database for later use.

2.3 *Open Data Service* (ODS)

2.3.1 *Open data* usability obstacles

We have so far discussed the concept of *open data*, then explained a bit about ETL processes. In the search for a clear definition of the concept of *open data*, we have explored some *open data* quality evaluation schemes and metrics. Researchers and engineers have came up with these metrics and schemes surely because the nature of *open data* makes it prune to irregularities and contamination more than closed data (Robinson & Scassa, 2022). These irregularities hinder the usability and reliability of *open data* as it requires great effort to fix and mediate the data before it is usable. Vetro et al. (2014) carried out an exploratory empirical assessment of the quality of OGD, which included conducting exploratory surveys administered to developers and *open data* consumers. These surveys aimed at exploring the common issues developers and data consumers face when dealing with *open data*, specifically OGD. The results of the surveys showed that *open data* consumers commonly face problems related to the following categories:

- Completeness issues
  - Data has missing values.
  - Data has incomplete or missing indices.
- Format issues
  - Data format is difficult to parse.
  - Data format is not open.
  - Data format needs to be changed otherwise the data is not usable.
- Traceability issues
  - Data lineage information are missing or insufficient.
  - Data versioning information are missing or insufficient.
- Congruence issues
  - Inconsistent data representation, for example, multiple ID schemes are used within the same data.
  - Data values inconsistent with declared domain, for example, values in a column fall outside the column domain.
- Heterogeneity issues
  - Data comes in heterogeneous chunks that differ in format or schema.
- Currentness issues
  - Data is not published as soon as it is available.
  - Data is not up-to-date.
- Understandability issues
  - Data has missing or incomplete metadata.

  - Data has missing or incomplete documentation.
  - Data has poor documentation and requires extra time and effort to understand its content.

- Accuracy issues
  - Data has incorrect or misaligned values.
  - Data contains misspellings.
  - Data has aggregation errors.
  - Data contains invalid values, for example, a negative length.

2.3.2 The cost of fixing *open data*

In order for *open data* to be usable in real-world applications, it has to undergo comparatively exhaustive cleaning and preparation processes. Survey results from a 2016 data science survey showed that data scientists spend nearly 80 percent of their work time on collecting, cleaning, and organizing data (CrowdFlower, 2016). Figure 2.1 shows a graph depicting the average allocation of work day time of a data scientist. The graph highlights the impact of data quality issues on data scientists' productivity as they spend most of the work day carrying out activities that precede the actual beneficial use of the data.

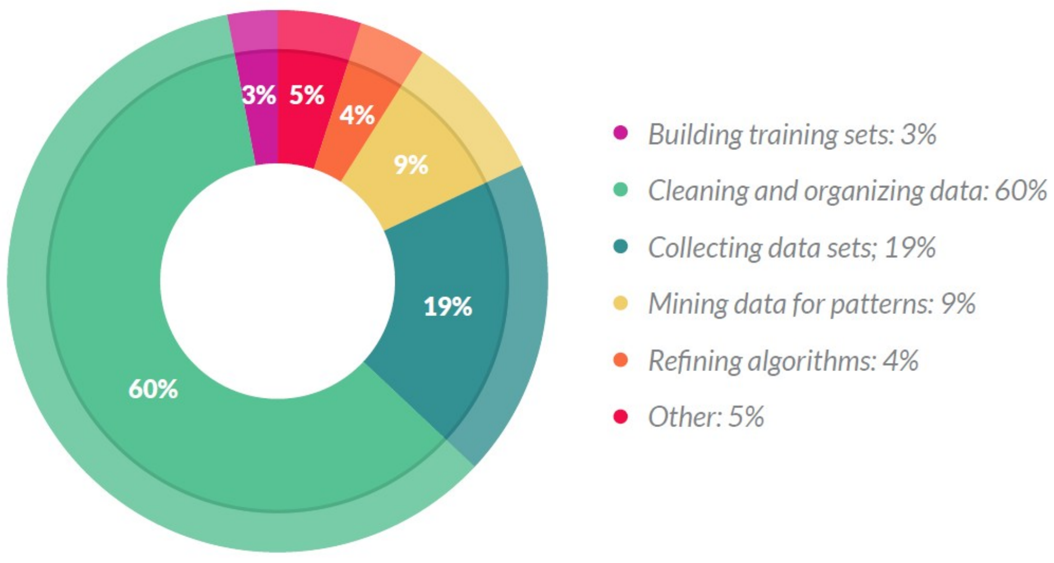

Figure 2.1: What data scientists spend the most time doing (CrowdFlower, 2016)

Closed data can have the same quality issues, but the fact that it is produced and published by a single dedicated source compensates for the cleaning efforts it exhausts, as the potential for consistency is much higher. Closed source data cleaning pipelines tailored for closed data sets are created within the organization that needs to use the data. Consequently, there will be no need for that organization to recreate a cleaning pipeline for the same data source. In addition, collaboration within same-organization teams is easy and frequent. Teams within an organization usually share the same data warehouse and can benefit from each other's assets and knowledge, which reduces the effort needed to reuse a data source that has quality issues. We also need not to overlook the fact that closed data sets are usually of high economic value, which makes spending time cleaning it a reasonable investment given the return. *Open data*, to the contrary, is usually published by public bodies and entities and reused by different data consumers for different applications. Collaboration is not guaranteed or organized between *open data* consumers of a certain *open data set*. In fact, collaboration on *open data* publishing -let alone consumption- is hard-to-attain, for example, some public entities have multiple departments and each may publish their own data without collaboration on a unified standard. This may have to do with the nature of *open data* initiatives, which makes investments in technical departments to support and coordinate data publishing processes undesirable as there is virtually no direct return (Concilio & Molinari, 2021).

> Government data is usually incomplete, out of date, of low quality, and fragmented. In most cases, *open data* catalogues or portals are manually fed as the result of informal data management approaches. Procedures, timelines, and responsibilities are frequently unclear among government institutions tasked with this work. This makes the overall *open data* management and publication approach weak and prone to multiple errors ('Global Report | Open Data Barometer', 2017, p. 14).

Concilio and Molinari (2021) referred that problem to a contradiction between data openness and its market value, which caused a market failure that requires government intervention. As a solution to this problem, the study suggested "incentivizing the creation and maintenance of *open* datasets" through government intervention in the form of:

> (a) direct subsidies to governments engaged in disclosing and maintaining their own datasets clean and accessible over time, or (b) new laws or regulations that impose the establishment of more productive data ecosystems, rewarding knowledge creation rather than mere data ownership (Concilio & Molinari, 2021, p. 10).

The solution suggested by Concilio and Molinari (2021) may not be applicable or widely adopted before a long time, and its application does not necessarily guarantee the desired results. This indicates that resolving *open data* usability obstacles is difficult to overcome due to absence of financial incentive and the distributed nature of the problem.

2.3.3 A solution from the *open source* world

Open source software initiatives have resulted in many innovations in the way people can collaborate on creating products without necessarily having the financial incentive to do so (Riehle, 2019b). Also, the nature of open source collaboration makes it very applicable to such distributed and large-scale problem. Open source software projects showed immense capabilities in attracting volunteer developers at a large-scale while collectively steering each other's efforts towards committing and accomplishing the goals of these projects (Riehle, 2011). This is due in part to the culture inherent to open source initiatives as "any potential volunteer could become a valuable resource. Thus, an effective project process must be open to accepting volunteers (egalitarianism), must recognize quality regardless of the source (meritocracy), and allow processes to develop according to the needs of the community (self-organizing)" (Riehle, 2015).

ODS was developed as a response to the needs of *open data* consumers for a streamlining interface between their data applications and *open data* providers, with an aim to create a community, in the spirit of open source collaboration, to crowdsource *open data* cleaning, curating, and adaptation efforts (Schwarz, 2019). The mission that ODS was set to accomplish was "to make consumption of *open data* easy, reliable, and safe" through "decoupling consumers from curators from publishers" so that collaborative innovation on using *open data* and fixing its quality issues becomes easier and faster (Riehle, 2019a). ODS provides the necessary structure for *open data* consumption, while enabling reusability of curated data configurations through wide adoption and community building, which is facilitated by the open source *AGPLv3* licensing of its core components.

2.4 *Data Science*

The term *data science* is used interchangeably with a lot of other terms to reference "the use of scientific methods and techniques, to extract knowledge and value from large amounts of structured and/or unstructured data"(Martinez et al., 2021 p. 4). These sets of activities have been referred to using other terms different than data science, for example, it was the term *data mining* that was commonly used to refer to *data science* activities until it was gradually replaced by the term *data science* during the past twenty years (Martinez-Plumed et al., 2021). There are other terms that refer to certain activities within the *data science* domain, such as: data analysis, data analytics, advanced analytics, deep analytics, descriptive analytics, predictive analytics, prescriptive analytics, data knowledge discovery, and data mining, which are highly connected and easily confused (Cao, 2017).

3 Requirements Engineering

Designing an application with the purpose of evaluating an ETL tool (ODS) fitness for use in a *data science* context requires unfolding the involved concepts and processes. We start this chapter by investigating the desirable qualities of an ETL tool supporting *data science* processes and activities. Then, we move forward to model ODS current features and characteristics. After that, we conclude the requirements for a *data science* application that can be used to accomplish the evaluation process.

3.1 Desirable Qualities of an ETL Tool for *Data Science*

3.1.1 Data science process and activities

Surveys in 2002, 2004, 2007, 2014, and 2020 showed a nearly unchanged prevalence of CRoss-Industry Standard Process for Data Mining (CRISP-DM) as the most popular data science process (Saltz, 2022). The surveys also showed that a variety of other processes are popular for data science projects execution. We throw light on some of the most widely adopted processes in apendix A. As CRISP-DM is the most popular *data science* methodology, we lay out its *phases*, *tasks*, and *outputs* in detail as follows:

phase Business understanding

task Determine business objectives

output Background

output Business objectives

output Business success criteria

task Assess situation

output Inventory of resources

output Requirements, assumptions and constraints

output Risks and contingencies

output Terminology

output Costs and benefits

task Determine data mining goals

output Data mining goals

output Data mining success criteria

task Produce project plan

output Project plan

output Initial assessment of tools and techniques

phase Data understanding

task Collect initial data

output Initial data collection report

task Describe data

output Data description report

task Explore data

output Data exploration report

task Verify data quality

output Data quality report

phase Data preparation

task Select data

output Rationale for inclusion/exclusion

task Clean data

output Data cleaning report

task Construct data

output Derived attributes

output Generated records

task Integrate data

output Merged data

task Format data

output Reformatted data

output Dataset

output Dataset description

phase Modeling

task Select modeling technique

output Modeling technique

output Modeling assumptions

task Generate test design

output Test design

task Build model

output Parameter settings

output Models

output Model description

task Assess model

output Model assessment

output Revised parameter settings

phase Evaluation

task Evaluate results

output Assessment of data mining results with respect to business success criteria

output Approved models

task Review process

output Review of process

task Determine next steps

output List of possible actions

output Decision

phase Deployment

task Plan deployment

output Deployment plan

task Plan monitoring and maintenance

output Monitoring and maintenance plan

task Produce final report

output Final report

output Final presentation

task Review project

output Experience documentation

3.1.2 Model of a highly performant ETL for *data science*

We have explored different research efforts aiming at outlining quality measures, defining evaluation criteria, eliciting requirements, and creating modeling and design techniques for optimum ETL processes. An elaborated review of the literature addressing these issues is laid out in appendix B. Through our review of the literature that addressed ETL processes quality from different perspectives, we could form an ensemble of the desirable qualities of ETL systems with a focus on supporting the *data science* activities and processes outlined in section 3.1.1. In this section, we expand on the inferred desirable qualities and features which constitute our model for optimum ETL for *data science*.

The overall structure of the model in the list below draws mainly from the work on ETL quality criteria in (Simitsis et al., 2009), (Theodorou et al., 2014), and (Theodorou et al., 2015). The bullets marking the list items include abbreviations for the purpose of categorization and organization. These are L, which stands for *level*, Q, which stands for *quality*, and DF, which stands for *desirable feature*. Each higher level quality is desirable in itself, and entails other desirable qualities and features. The degree of absence or existence of the underlying qualities and features is decisive for assessing an ETL tool's fitness for serving data applications or supporting data science activities and processes. The second-level qualities are essential for achieving the higher level ones, and are desirable in themselves as well. There are certain measures, mentioned throughout the literature on ETL evaluation criteria, that can be used to approximately quantify the degree of absence or existence of the above mentioned qualities. These measures might be useful for comparison purposes, but they might be less useful for evaluation of a single ETL tool. The underlying *desirable features* in the list are special features that have been widely brought up and recommended throughout the literature that we reviewed during our research, which were mainly focused on ETL process quality and evaluation criteria. The concluded modelling of a highly performant ETL process comprises the qualities, characteristics, and features listed below.

L1Q1 *data quality*

- L2Q1 *data accuracy*
- L2Q2 *data completeness*
- L2Q3 *data freshness*
- L2Q4 *data consistency*
- L2Q5 *data interpretability*
- DF01 *schema mapping* capabilities
- DF02 ability to define inter-attribute relationships
- DF03 *data cleansing* capabilities
- DF04 variable update cycles
- DF05 *data profiling* capabilities
- DF06 *entity recognition* and *matching* across sources
- DF07 *data enrichment* capabilities
- DF08 *change data capture* capabilities
- DF09 *incremental update* capabilities
- DF10 Relational Online Analytical Processing (ROLAP) or Multidimensional Online Analytical Processing (MOLAP) capabilities
- DF11 ability to fetch, define, or accommodate data documentation

L1Q2 *performance*

- L2Q6 *time efficiency*
- L2Q7 *resource utilization*
- L2Q8 *capacity*
- L2Q9 *supported modes*
- DF12 ability to handle large number of sources or pipelines concurrently

L1Q3 *security*

- L2Q10 *confidentiality*
- L2Q11 *integrity*

L2Q12 *reliability*

L3Q1 *availability*

L3Q2 *fault tolerance*

L3Q3 *robustness*

L3Q4 *recoverability*

L3Q5 *redundancy*

DF13 understandable and actionable error and diagnostics reports

DF14 possibility of *low-code* or *no-code* debugging

DF15 rich debugging features

DF16 quick recovery from failure

DF17 *re-entrant* processes

L1Q4 *auditability*

L2Q13 *traceability*

DF18 advanced *metadata* management

DF19 detailed *data lineage* documentation and reporting

DF20 ability to produce *metadata* reports

DF21 ability to produce *impact analysis* reports

DF22 ability to produce *data lineage* reports

DF23 *metadata* interfaces for querying and editing

L1Q5 *adaptability*

L2Q14 *scalability*

L2Q15 *flexibility*

L2Q16 *reusability*

L2Q17 *extensibility* (adding and integrating user-defined functionality)

DF24 reusable procedures

DF25 inter-component ETL processes capabilities (multi-faceted usage)

DF26 integrations with third-party tools and suites

DF27 ability to connect to different types of data sources

DF28 easy integration of new data sources

DF29 smart execution based on predefined conditions

DF30 intelligent adapter that can connect to different data stores in different formats

L1Q6 *usability*

L2Q18 *understandability*

L2Q19 *cost efficiency*

L2Q20 *openness*

L2Q21 *ease of use*

DF31 possibility of *low-code* or *no-code* operation and management

DF32 enhanced graphical development

DF33 reduced need for user-written procedures

DF34 *no-code* or *low-code* transformations, at least for the common ones

DF35 data scientists friendly *transformation* language

DF36 powerful and rich *transformation* language

DF37 *no-code* or *low-code schema mapping*

DF38 visual mapping interface

DF39 easy and fast deployment

DF40 permissive *licensing*

DF41 low *cost of ownership*

DF42 active support

DF43 large community of users

DF44 up-to-date and extensive documentation

L1Q7 *manageability*

L2Q22 *maintainability*

L2Q23 *testability*

DF45 *open-source* code for core components

3.2 Current Qualities of ODS

ODS has gone through multiple development iterations, and is still undergoing many enhancements and developments. Thus, we will not follow an inside-out approach focusing on under-the-hood technical features of the current implementation of ODS. Instead, we will follow a user-centric approach, where the features that will be modeled are those that the user, or more precisely the *data scientist*, can interact with. We may also bring up technical or architectural features that are of relevance from the perspective of the *data scientist* user. An illustration of the current architecture of ODS and hints on some of the technology choices made in the implementation are provided in figure 3.1.

Examining the GUI and the API of the ODS provides a clear view of the scope of functionality that the ODS can support as an ETL. An elaborated examination of the ODS GUI and API functionality is laid out in appendix C. A remarkable observation is that ODS goes beyond providing ETL functionality by the inclusion of integrated data warehousing capabilities. The ODS has an integrated *PostgreSQL* database for storage of processed data. Remarkably, it also employs *Liquibase* database version control system for tracking of database schema changes, which provides some sort of *data lineage* documentation. The database solution is wrapped using *PostgREST* so that it is accessible through its own *RESTful API*. Table 3.2 demonstrates a model of ODS features supporting each phase of its ETL functionality as well as its *data warehousing* functionality and general features.

3.3 Requirements for an Evaluation Application

We may now proceed to lay out the requirements for a data science application that can be used to evaluate ODS performance as an ETL. We will depend on the two models we have created: the model for a highly performant ETL for data science, and the model of current ODS qualities and features. We will aim at engineering the requirements in a way that makes the application touch upon most of the quality criteria and desirable features mentioned in the model for a highly performant ETL for data science. However, we will try to adhere to the outlines of the model of the current capabilities of ODS, as features that are not feasible through the ODS will be out of the scope of the evaluation process.

The requirements are laid out in the list below. The list has a three-level hierarchy. Requirements are listed in the highest level, with bullet labels in the following style: <{F/NF}R{serial number}>, where F stands for *functional*, NF stands for *non-functional*, and R stands for requirement. Each requirement statement implicitly starts with "the evaluation application should". The second

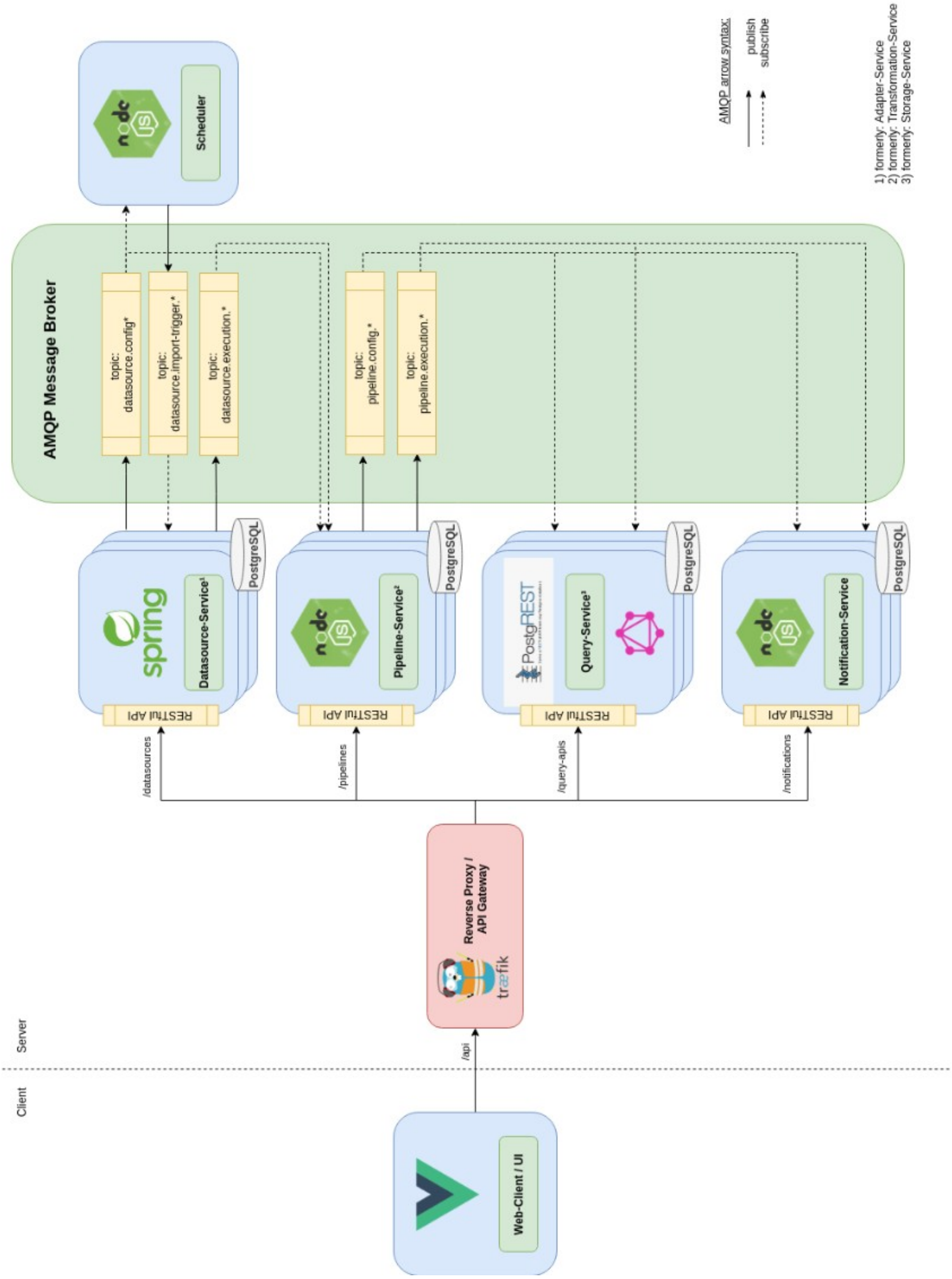

Figure 3.1. *Microservices* architecture of ODS v2 (Jvalue Project, 2022).

Table 3.1: Model of the ODS features allocation as an ETL.

General
Most features are available through a GUI
Most features are available through an API
Manageable through GUI alone
Manageable through API alone
Microservices architecture
Open-source license
Shareable *data source* configuration
Shareable *transformation* scripts
Integrated *data warehousing*

Extract	**Transform**	**Load**	**Warehousing**
Supports HTTP protocol	*Pipelines* management interface	Allows retrieval of data of a pipeline	Integrated *PostgreSQL* database
Allows raw *data imports*	*Pipelines* configuration interface	Allows deletion of *storage structures*	Integrated *Liquibase* version control
Integrated *data view* of raw source data	Allows creating multiple *pipelines* for a single *data source*	Allows manually dumping data into a pipeline *storage structure*	Accessible through a *PostgREST*-enabled *REST API*
Connects to *JSON* sources	Scripted data transformation	Processed data is automatically loaded into the corresponding *storage structure*	
Connects to *CSV* sources	Live transformation testing	Allows creating *storage structure* for each pipeline	
Connects to *XML* sources	*Pipelines* metadata management	Connects to the integrated database via a *REST API*	
Metadata management interface	*Pipeline*-associated notifications		
Data sources configuration interface	Separate *notifications* management interface for each pipeline		
Data sources management interface	Supports Firebase *notifications*		
Supports batch processing mode	Supports Slack *notifications*		
Supports stream processing mode	Supports Webhook *notifications*		
Metadata display interface	Allows one-off *transformation* jobs		
Allows periodic data fetching	Integrated *data view* of processed data for each *pipeline*		
Adjustable periodic fetching intervals	Allows manually feeding data batches into a same-source pipeline		
	JavaScript *transformations* scripting		

level is dedicated for listing qualities from the model in section 3.1 that a certain requirement helps evaluate. The third level is dedicated for providing brief reasoning for including the corresponding requirement in the upper level.

FR1 consume data that has *data quality* issues

assesses L1Q1, L2Q1, L2Q2, DF01, DF03, L1Q3, L2Q12, L3Q3, L3Q4, DF13, DF16, DF17

∗ Low quality data is needed to assess the degree of *data quality* and *data accuracy* attainable through ODS.

∗ *Data consistency* issues might cause workflow crashes, and this will help assess features such as *reliability*, *robustness*, and *recoverability*.

FR2 consume *open data*

assesses L1Q1, DF03, L2Q20, DF27, L1Q3, L2Q12, L3Q3, DF16

∗ The main premise of ODS is about *open data* consumption.

∗ For reasons explained in section 2.3.1, *open data* is more likely to contain *data quality* issues which is needed to assess the degree of *data quality* attainable through ODS.

FR3 consume data that is mostly numerical

assesses L1Q1, L2Q1, L2Q2, L2Q5

∗ Residual *data accuracy* and *data completeness* issues are easier to detect in numerical data.

FR4 consume data with high-frequency publication intervals

assesses L2Q3, L2Q4, DF01, DF04, L1Q2, L2Q6, L2Q8, DF12, L2Q12, L2Q14, DF29

∗ Consuming data from sources that frequently push new instances helps assess the level of *data freshness* attainable through ODS.

∗ High-frequency data fetching will help evaluate *performance*, *reliability*, data update cycle, and conditional execution aspects.

FR5 require multi-access and usage of output data

assesses L2Q4, L1Q2, L2Q5, L2Q6, L2Q7, L2Q8, L2Q11, DF27, L2Q18, DF39

∗ Multi-access of processed data is critical for evaluating ODS performance regarding *data consistency*.

  * Multi-access and retrieval of data helps test *resource utilization*, *capacity*, *integrity*, and *understandability* aspects of ODS.

FR6 require executing *data transformation* pipelines

assesses DF01, DF03, L2Q8, DF21, DF24, L1Q6, DF34, DF35, DF36, DF37, DF38, L1Q7

  * *Data transformation* taps on most aspects of the ETL functionality of ODS.
  * Including such complex procedure is critical for evaluating *ease of use* and *manageability*.

FR7 require metadata management

assesses DF11, L1Q4, L2Q13, DF18, DF20, DF23, DF32, L1Q7

NFR1 exert adequate level of load on ODS services

assesses L1Q2, L2Q6, L2Q7, L2Q8, L3Q5, L2Q14

  * Implicit and unexpected performance shortcomings are more likely to be discovered under heavy load volumes.

NFR2 require very low response times

assesses L2Q3, L1Q2, L2Q6, L2Q9

  * Response time is decisive in assessing an ETL ability to support *stream processing*.

NFR3 require high throughput rates

assesses L1Q2, L2Q6, L2Q8, DF12, DF39

FR8 require high degree of *data freshness*

assesses L2Q3, DF04, L1Q2, L2Q6

FR9 require notification of readiness of new data

assesses L2Q3, DF04, L1Q2, L2Q6, L1Q4, DF26, L1Q6, L2Q21, DF31, L1Q7

  * Setting up notifications for data readiness is important for assessing integration capabilities of ODS.

FR10 require varying data fetching intervals

assesses L2Q3, DF04, L1Q2, L2Q6, L2Q6, L2Q15, DF27, DF28, L1Q6

  * Varying data fetching intervals will help evaluate *flexibility*, *manageability*, *ease of use*, and smart execution features.

FR11 consume data form multiple data sources

assesses L1Q1, L2Q1, L2Q2, L2Q3, L2Q5, DF01, DF03, DF04, L1Q2, L2Q8, DF12, DF18, DF23, L1Q5, L2Q14, DF27, DF28, DF30, L1Q6, L2Q21, DF36, L1Q7

∗ Connecting to multiple data sources will create a more realistic set-up that benefit all evaluation activities.

∗ Multi-source data consumption will help thoroughly evaluate the *performance* and richness of *data transformation* part of ETL functionality of ODS.

FR12 require multiple different transformation pipelines

assesses L1Q1, L2Q4, L2Q5, DF01, DF03, L1Q2, L2Q8, DF12, L2Q14, L2Q21, DF36, DF39, L1Q7

∗ Real-world *data science* process requires multiple different *data transformation* pipelines to support different data usage scenarios.

FR13 require one-off data processing jobs

assesses DF01, DF03, DF04, L1Q2, L2Q8, L2Q9, L1Q5, L2Q15, DF27, DF27, DF30, L2Q21, L1Q7

∗ One-off data processing jobs are usually needed at the start of *data analysis* projects to retrieve historical data batches. Even in *stream processing* usage scenarios, it precedes the processing of data streams.

∗ Processing one-off data batches will help assess *capacity*, *data cleansing*, *data transformation*, and *manageability* aspects of ODS.

FR14 require frequent and concurrent access to processed data

assesses L2Q4, L1Q2, L2Q6, L2Q7, L2Q8, DF12, L1Q3, L2Q11, L2Q12, L3Q1, L1Q6

∗ Access concurrency is critical for evaluating *data consistency*, *confidentiality*, *integrity*, as well as *performance* aspects.

FR15 require re-use of transformation scripts

assesses DF01, DF03, L1Q5, L2Q16, DF24, DF28, L2Q21, DF33, DF35, DF39, DF40, L1Q7, L2Q22

FR16 require live transformation testing

assesses DF01, DF03, L2Q6, L1Q3, L2Q11, L1Q6, L2Q21, DF35, DF36, L1Q7, L2Q23

FR17 require integration of workflow management within application code

assesses L1Q1, L2Q10, L2Q11, L1Q5, L2Q15, L2Q16, L2Q17, DF24, DF26, L1Q6, L2Q20, L2Q21, DF39, DF40, DF44, DF45

∗ Requiring programmatic workflow management is essential for evaluating *extensibility*.

FR18 require workflow monitoring through GUI

assesses L1Q4, DF23, L2Q15, L1Q6, L2Q18, L2Q21, DF31, DF32, DF34, DF37, DF38

∗ Workflow monitoring through GUI is a major criteria for evaluating *ease of use* and *understandability* (Wayne Eckerson & Colin White, 2003).

FR19 require access to ODS source code

assesses L1Q4, DF15, L2Q17, L2Q18, L2Q20, DF40, DF44, DF45

FR20 require access to ODS documentation

assesses L1Q6, L2Q18, L2Q20, L2Q21, DF39, DF44

FR21 require very low cost of ownership

assesses L1Q6, L2Q19, DF39, DF40, DF41, DF45

The model in section 3.1 stresses seven high-level qualities, 23 underlying qualities, and 45 features and characteristics that are highly desirable for an ETL for data science. The above requirements touch upon most of the qualities and features highlighted by the model. However, there are some qualities and features that can not be evaluated due to lack of support in the current implementation of ODS. For example, *data lineage* and *impact analysis* reporting capabilities can not be assessed as those are not yet supported through ODS metadata management interfaces. There are some qualities and characteristics that can not be evaluated through application requirements, such as DF43 (large community of users). However, this does not prevent the evaluation of these characteristics using other suitable measures.

3.4 Selection of an Evaluation Application

3.4.1 Guiding principles

Based on the requirements laid out in section 3.3, the *data science* process specifications in section 3.1.1, and the model of ODS v2 capabilities in section 3.2, we can form a clear conception of candidate evaluation applications. In addition, there are some implicit requirements that should be taken into consideration. The evaluation application has to be non-trivial in order to cover a wider scope of users (*data scientists*) needs. It also has to resemble the endeavours and projects that are common in real-world *data science* settings. In other words, it shouldn't be off the beaten track. *Data science* online collaboration platforms represent a window on the current problems and questions that *data science* projects are set to solve. The widely popular *data science* competitions platform, *Kaggle*, can provide important insights on the main paths that *data science* projects take. A sample from the list of available competitions on *Kaggle*, in descending order according to the total number of competing teams, is shown in table 3.3.

Competition	Nr. of teams
Santander Customer Transaction Prediction	8751
Home Credit Default Risk	7167
Forecasting of *Walmart* Unit Sales	5558
Toxic Comment Classification Challenge	4539
COVID-19 mRNA Vaccine Degradation Prediction	1636

Table 3.3: Sample of Kaggle competitions with the highest number of competing teams (Kaggle Inc., 2022).

3.4.2 Viable candidates for evaluation

Drawing from the aforementioned resources, we can consider the following *data science* projects as viable candidates for an evaluation application:

CA1 A *data analysis* project aiming at creating *business insights* reports and providing data exploration and visualization interface. The application is intended for in-house use for a company in the energy sector. The application enriches internal business data using OGD in order to provide richer context. The application has two interfaces. The first is dedicated for on-demand *business insights* reports generation. The second interface is for real-time visualization and exploration of enriched business data. The application depends on two data sources: internal business data; and energy OGD.

CA2 A *time-series* forecasting project aiming at prediction of prices per square feet of housing in each state in the United States. The application outputs a prediction of next month prices per square feet for each US state. It depends on two distinct but related sources of data: aggregate batches of historical data; and new housing prices data that is published monthly.

CA3 An *online learning time-series* forecasting project aiming at prediction of COVID-19 statistics. The application outputs forecasts of COVID-19 deaths, cases, vaccination rates, and recovery rates after ingesting the latest published statistics. It depends on eight sources of data: historical COVID-19 deaths data; daily COVID-19 deaths data; historical COVID-19 cases data; daily COVID-19 cases data; historical COVID-19 vaccination rates data; daily COVID-19 vaccination rates data; historical COVID-19 recovery rates data; and daily COVID-19 recovery rates data. The project employs a separate model for forecasting each of the four statistics. The models use new data instances to update its parameters and provide better forecasts.

CA4 A *classification predictive modelling* project aiming at live prediction of win probability of teams in ongoing football games. The application provides updated estimates for win probability of each team during the match using game statistics and players data as its input. The project employs a *machine learning* model that updates its predictions on-the-spot as new data arrives. It depends on three sources of data: historical football game statistics and player data for initial training; *data streams* of game statistics to provide in-game predictions; and post-game data that is used for model validation and update.

3.4.3 Comparison and selection

We now compare the four viable candidates in order to select an evaluation application to implement. The comparison is carried out against the fulfilment of the requirements laid out in section 3.3. Scores and results of the comparison are listed in table 3.4. The comparison results shows that the second candidate (CA2) has an advantage over other candidates, as it has mostly checked all the boxes. The fourth candidate fulfills most of the requirements, but it falls short on some of the most critical requirements. For example, football game statistics data is usually collected, aggregated, and made *open* by *sport analytics* companies, which makes the likelihood of frequently running into *data quality* issues very low. Consequently, we proceed with implementing the third candidate (CA3) to evaluate ODS performance as an ETL.

Req. ID	Candidates			
	CA1	CA2	CA3	CA4
FR1	✓	✓	✓	
FR2		✓	✓	✓
FR3	✓	✓	✓	✓
FR4			✓	✓
FR5	✓		✓	
FR6	✓	✓	✓	✓
FR7	✓	✓	✓	✓
NFR1	✓		✓	✓
NFR2	✓		✓	✓
NFR3	✓		✓	
FR8	✓		✓	✓
FR9	✓		✓	✓
FR10	✓			✓
FR11	✓		✓	✓
FR12	✓		✓	✓
FR13	✓	✓	✓	✓
FR14	✓		✓	✓
FR15			✓	✓
FR16	✓	✓	✓	✓
FR17		✓	✓	✓
FR18	✓		✓	
FR19	✓	✓	✓	✓
FR20	✓	✓	✓	✓
FR21		✓	✓	

Table 3.4: Comparison of the four evaluation application candidates on the basis of requirements fulfillment.

4 Architecture, Design, and Implementation

In this chapter, we will expand on the architecture, design, and implementation of the chosen candidate for evaluation application. We start by discussing architectural and design decisions that shaped the implementation. We then move forward to discuss the implementation and highlight some of the problems that we encountered and the solutions we applied to resolve it. It is important to keep in mind that the evaluation application is not the focus or the goal of this study in itself. It is a means to an end, which is evaluating ODS performance as an ETL in a *data science* setting. As a result, we will address the aforementioned phases in a brief manner.

The definitions of the terms *architecture*, *design*, and *implementation* overlap frequently, and it is not straightforward to address each of these categories with clear-cut distinction (Eden & Kazman, 2003). The work in (Eden & Kazman, 2003) suggests using certain criteria to distinguish these concepts. The study suggests using *intension* and *locality* as the main characteristics by which *architecture*, *design*, and *implementation* can be differentiated. *Intension* and *locality* both describe the abstraction of each of the three types of specifications. The two attributes were defined in (Eden & Kazman, 2003) as:

- *Intensional* specifications are conceptual, or "can be formally characterized by the use of logic variables" (Eden & Kazman, 2003, p. 2).
- *Non-local* specifications apply to the whole system, not to a specific part.

By this definition, according to (Eden & Kazman, 2003), *architecture* can be considered as specifications that are both *intensional* and *non-local* ; *design* specifications can be considered those that are *intensional* but *local* ; and *implementation* specifications can be considered those that are *extensional* and *local*. In the following sections, we will try to discuss aspects of the evaluation application software with an approximately correct allocation under these categories..

4.1 *Architecture*

The architecture of the evaluation application was developed to achieve three goals:

- Carry out project scenario mentioned in section 3.4.2.
- fulfill requirements laid out in section 3.3.
- utilize as much ODS features as possible.

In addition, the project scenario indicates that the application will result in four different *online learning time-series forecasting models*. For that, the architecture needed to support two modes of operation:

- Initial training mode
- Online learning mode

ODS acts as an ETL that extracts raw data from *data publishers*, processes it, and then loads it into a *data warehouse*, which is then queried to provide the processed data to train and update the models. Thus, the evaluation application employed a layered architecture as it is more suitable to achieve the modularity needed to accomplish the aforementioned goals, support different operation modes, and accommodate the different phases within the application. Figure 4.1 shows a diagram of the architecture of the evaluation application.

4.2 *Design*

In order to fulfill the requirements through the aforementioned architecture, the application was designed as follows:

- *Data Publishers*:
  - Composition:
    - Four data publishers.
    - Eight data sources: four for one-off historical data batches, and four for data streams.
  - Interactions:
    - Data publishers with APIs receive requests with dynamic parameters from ODS.
    - Data publishers with APIs respond to data fetching requests and send the data to ODS in response.

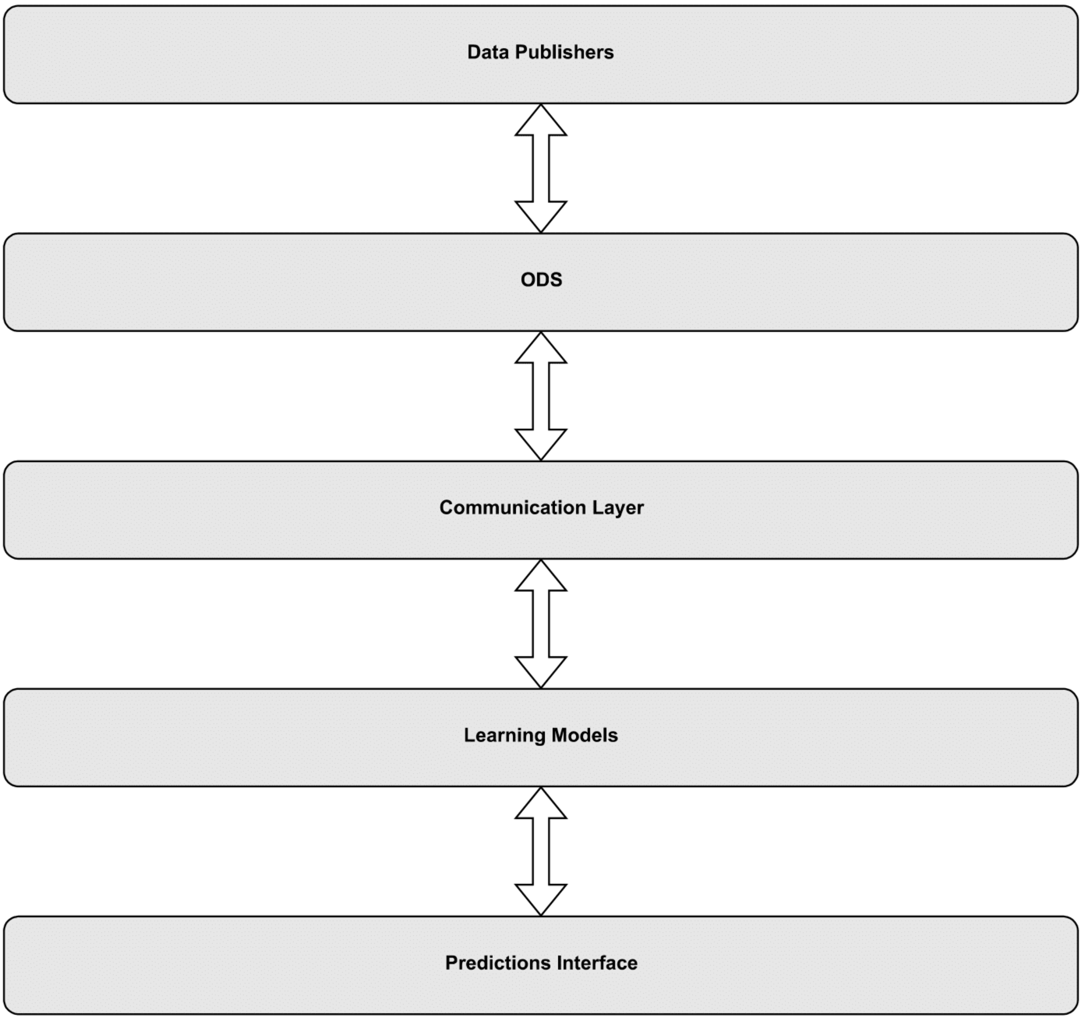

Figure 4.1: *Architecture* of the evaluation application.

    ∗ Data sources with no APIs are fetched as a whole.
    ∗ Data sources receive periodic data fetching requests matching their data publishing intervals.

- *ODS* :
  - Composition:
    ∗ The services exposed through the API and the GUI are used. These are *query* service; *datasource* service; *notification* service; and *pipeline* service.
  - Interactions:

        * ODS *datasource* service sends data fetching requests according to respective data source configuration.
        * ODS *datasource* service receives responses or fetched data and passes it on to services downstream.
        * ODS receives configurations of *data sources*, *pipelines*, and *notifications* and implements them.
        * ODS receives data fetching requests and provides the requested data in response.
        * ODS receives data transformation requests and provides the processed data in response.
        * ODS sends notifications of readiness of pipelines output to *Learning Models* module through *Communication Layer*.

- *Communication Layer* :
    - Composition:
        * *Notification Medium*.
        * *ODS API Client*.
    - Interactions:
        * *Notification Medium* receives data readiness notifications from ODS.
        * *Notification Medium* triggers the models update cycle.
        * *ODS API Client* sends configurations of *data sources*, *pipelines*, and *notifications* to ODS.
        * *ODS API Client* sends data fetching and data transformation requests to ODS, receives the data in response, and makes it available for consumption by *Models Training* module.
        * *ODS API Client* sends requests for *pipelines* and *data sources* metadata information, receives it, and makes it available for processing by *Models Training* module.
- *Models Learning*:
    - Composition:
        * *Data Streams* module, which contains four data streams.
        * *Models* module, which contains four models.
    - Interactions:

    * *Data Streams* module passes configurations of *data sources*, *pipelines*, and *notifications* to ODS through *ODS API Client*.
    * *Data Streams* module sends data fetching and data transformation requests to *ODS API Client*, receives the data in response, and passes it to *Models* module.
    * *Data Streams* module receives information needed for fetching new data from *Notification Medium*, after *Notification Medium* parses the notification of readiness it received.
    * *Data Streams* module passes processed data to *Models* module.
    * *Models* module receives processed data from *Data Streams* module in order to train or update its models.
    * *Models* module receives requests for predictions from *Predictions Interface*, and sends the predictions in response.

- *Predictions Interface*:
  - Composition:
    * "Messenger" service.
    * "Interface".
  - Interactions:
    * "Messenger" service sends requests for predictions to *Models* module.
    * "Messenger" service receives predictions, and passes it on to *Interface*.
    * "Messenger" service receives parameters for predictions requests from *Interface*.
    * "Interface" sends parameters for predictions requests to *Messenger*, and receives the predictions in response.

Through the design laid out in the above list, the application is able to fulfill the requirements using the chosen architecture, while being able to switch operation modes smoothly. For *initial training* mode, this design allows the application to fetch one-off historical data batches from data sources, execute transformation jobs outside pipelines, fetch resulting processed data, and finally train models. For *online learning* mode, this design allows the application to configure data sources for periodic fetching according to predefined intervals, create *data transformation* pipelines, receive notifications of readiness of processed data, fetch the new data, update the models, and receive requests for predictions, and finally provide predictions to be displayed in the predictions interface. A diagram of the aforementioned design is shown in figure 4.2.

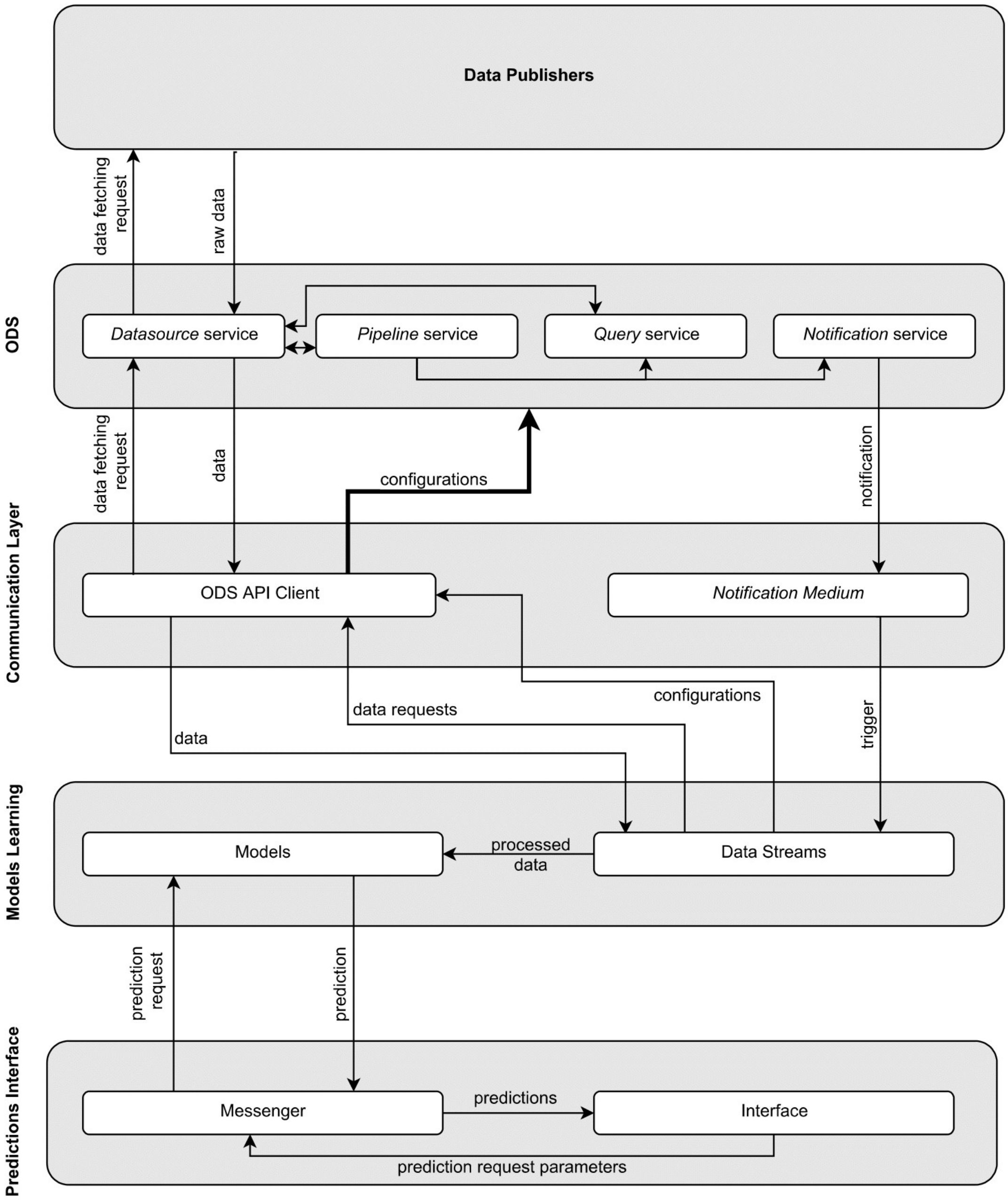

Figure 4.2: *Design* of the evaluation application.

4.3 *Implementation*

The implementation of the evaluation application was carried out in accordance with the requirements laid out in section 3.3, the architecture laid out in section 4.1, and the design laid out in 4.2. We will discuss the implementation aspects of the evaluation application in the same order as the layers of the architecture. However, we will start by discussing general implementation measures that needed to be taken in order to fulfill requirements that cannot be satisfied by a single module.

The source code of the application is written in *Python*, as many *Python* libraries were needed for downstream *machine learning* and *data manipulation* tasks. For example, *pandas* library was employed during the implementation as it provides powerful intermediary data structures and containers, namely *data frames*, to contain the incoming data from *ODS* and make it easily consumable by *machine learning* libraries. Additionally, the API client for ODS is written in Python. The evaluation application was deployed on a machine that operates *Windows* operating system, in addition to *Windows Subsystem for Linux v2*. Throughout the implementation, the choice of software components was restricted to free and *open source* software to fulfill requirements such as FR19 and FR21. *Git* was used for version control of the application source code, and the repository was hosted locally, and remotely on *GitHub*.

For *Data Publishers* layer, data sources varied between *GitHub* repositories of *Robert Koch Institute*, an API provided by *Robert Koch Institute*, and *time-series* data hosted on *ArcGIS* and provided through its API. The sources had varying publishing intervals, but this could be overcome by setting the interval to occur well past the variation range. The formats varied between *JSON* and *CSV*. The data sources were chosen so that the data was proven to have *data quality* issues, be mostly numerical, and be *open*. In addition, ODS capabilities were taken into consideration, as the formats and protocols used to convey data were all supported in the current version of ODS.

Version two of the ODS was used in the implementation, as it is the subject of this study. The deployment of ODS was triggered using the command-line interface of the ODS API client. ODS containers were mounted and the *Docker* system started the ODS application. The evaluation application initially went through crashes, and required frequent restarting and debugging, but ODS was left operating continuously, even when the evaluation application was not running, to test its *robustness* and *resilience to failures*. The deployment of ODS was easy and fast using its *Python* API client. Through the API client, we were able to start, stop, and reset ODS using a one-line command in the command-line interface. The evaluation application also tested programmatic deployment of ODS from within the application code using modules from ODS API client, and it was equally straightforward and efficient. ODS GUI was used throughout the evaluation application for monitoring, workflow management, live transformation testing, and auxiliary metadata management.

As the programming language for the implementation is *Python*, we chose the *Python* client of ODS API to be the main means of communication in *Communication Layer* between the rest of layers downstream and *ODS*. It provides access to the endpoints of ODS API, in addition to a set of curated features that are provided by ensembling functionality of different endpoints. It also makes conveying data and configurations throughout the evaluation application easier, as it uses *Python* objects and data structures. For *Notification Medium*, *webhooks* were chosen to carry out this role. ODS provides the possiblity of setting *webhooks*, *firebase*, or *Slack*, but we chose to proceed with *webhooks* as they can be set-up and managed programmatically from within the application without changing any other aspects of the implementation. *Flask*, and the accompanying libraries, were used to implement and deploy the *webhooks*.

Models Learning layer contains four different models. Each model requires a stable feed of processed data, and undergoes different stages of development from initial training to parameters update. Thus, Object-Oriented Programming (OOP) paradigm principles were applied throughout most of the implementation of *Models Learning* layer. A UML diagram of the *Data Streams* module is shown in figure 4.3. *Models* module also followed OOP principles, and was organized into five classes as shown in figure 4.4. A parent class, "Model", that has no custom "init" method, and four child classes that implement only an "init" method containing all the required parameters for training, tuning, and using the models. The initial training of the models required testing different types of *time-series forecasting* models and algorithms. To carry out the required experiments, *PyCaret* machine learning library was used to carry out complex and extensive experiments in an efficient and well-documented manner. In addition, to the *PyCaret* logs, experiments were also documented extensively using *MLflow* library. The complete set of logs and experiments documentations are included within the application code repository. For the final tuning, deployment, and update of the models, *sktime* machine learning library was used as it provides more low-level access to models functionality.

The last layer in the application is *Predictions Interface*. This layer contains two modules: *Messenger*, and *Interface*. As this layer does not test or assess any of ODS qualities, we did not introduce any uncessary complexity into its implementation. Simple *sockets* were used to enable *Messenger* module of sending

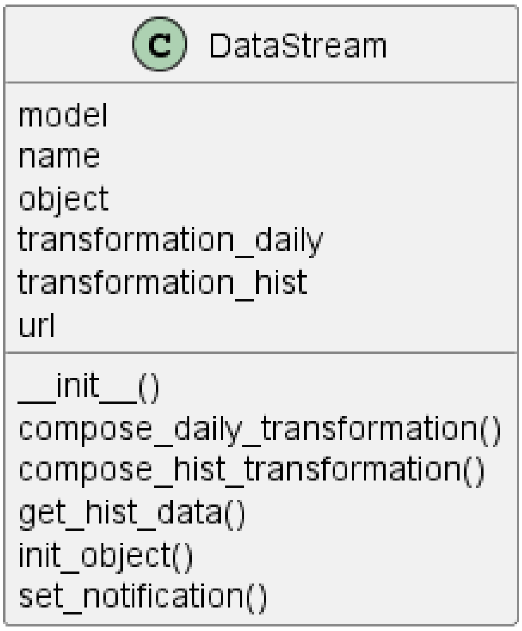

Figure 4.3: UML diagram of *Data Streams* module.

prediction requests and receiving predictions, which are then passed to *Interface* for display. *Interface* component was implemented in a way that employs the *command-line interface* (CLI) to receive input parameters for prediction requests and display the results. In addition to the CLI, the locally hosted web UI of *MLflow* was used to monitor models update and prediction results.

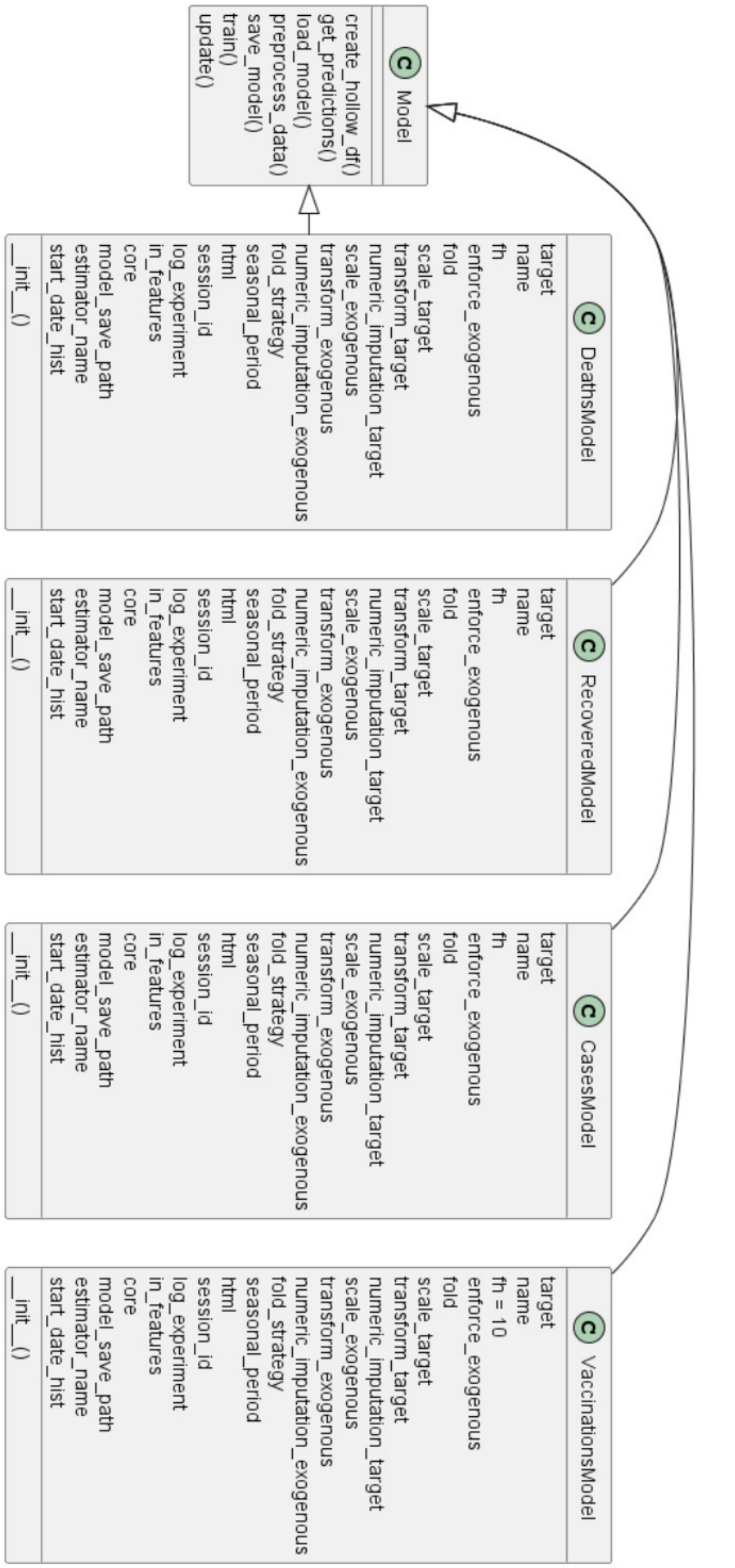

Figure 4.4UML diagram of *Models* module.

5 Results

5.1 Evaluation

During and after the implementation of the evaluation application, the evaluation process was carried out. During our review of literature on *ETL* desirable qualities and evaluation criteria, we encountered definitions of metrics that can be used to quantify an ETL tool performance regarding a certain quality. We will employ some of those metrics. However, we found some metrics to be taking an impractical approach towards quantification, which is useful for measuring performance in real-world settings. The evaluation results of ODS performance with respect to the quality criteria defined in the model in 3.1 are listed below.

A reasonable level of *data quality* was attainable through ODS. Fair levels of *data accuracy* and *data completeness* could be reached. However, it required tedious manual checks, complex scripting, and postprocessing to achieve it. In terms of *data freshness*, ODS performed well, thanks to high *throughput* and low response time. The evaluation application thoroughly tested *data consistency*, and it was clear that ODS could perform well in that regard. Due to the lack of advanced metadata display and management, and data documentation capabilities, ODS could only marginally improve *data interpretability*. *Schema mapping* was possible through ODS transformation scripting, but the process was lengthy, risky, and not suitable for real-world data-intensive applications that require contain large numbers of variables, attributes, data sources, and pipelines. The same evaluation is valid for the ability to define inter-attribute relationships, and for *data cleansing* capabilities. Through the evaluation, it was clear that ODS provides strong support for variable update cycles management using periodic fetching intervals. Regarding DF05, ODS does not provide any *data profiling* capabilities. The same evaluation is valid for the remaining desirable features under the high-level *data quality* criteria.

As the evaluation application consumes data from eight data sources, and requires execution of *data transformations* on all fetched data. This enabled evaluating *performance* quality criteria. The evaluation application entails *online learning* models that require high degree of *data freshness*. The required level of *data freshness* was reached thanks to the remarkable degree of *time efficiency* that the ODS provides. Regarding *resource utilization*, the loading and operation of ODS contributed the largest increase to *resource utilization* as *CPU* utilization went up by 18 percent, and memory utilization went up by 21 percent. However, ETL operations through ODS, especially during data fetching and processing, did not contribute large increases to *resource utilization*. Regarding *capacity*, ODS was able to fetch data, execute pipelines, and load data through the query service in with the same *time efficiency* at all levels of load. To further assess this criteria, periodic fetching intervals were set so that data fetching would be triggered for all data sources simultaneously. The results of this evaluation proved that ODS can maintain its level of *performance* and fulfill *time efficiency* and *throughput* requirements under high-load conditions. Regarding *supported modes*, the evaluation application was designed to exert load that resembles *data streaming* during periodic data fetching, and ODS could show low response times and high throughput under these conditions. This shows it may be possible to support *data streaming* mode in the current version of ODS. Our application employed consumed eight data sources, four of which were configured for periodic data fetching, executed eight transformation workflows, four of which were recurring data transformation pipelines, and ODS could support these activities with high *performance*, this proves it is able to handle large number of sources and pipelines concurrently which satisfies DF12.

As ODS is still not widely adopted, its deployment is confined to a narrow number of settings. In addition, ODS is mainly adopted for *open data* consumption where *confidentiality* and *integrity* qualities are not of high urgency. Thus, ODS does not provide rich *security* features or high level of *confidentiality* and *integrity*. For example, user authentication functionality seems to be suspended in ODS v2, let alone user roles and access privileges. Regarding *reliability* and *availability*, the current implementation of ODS is not platform or OS independent, and this affected its performance with respect to these qualities. For example, *data warehousing* problems and crashes arose when operating on *Windows* OS. During the evaluation, we inserted an incorrect transformation script into a pipeline to test error handling behavior of ODS. The pipeline could not be executed. However, ODS could resume normal operation on other pipelines and did not crash. This indicates a fair level of *fault tolerance* and *robustness*, and a low level of *recoverability*. Error and diagnostics reporting in ODS is very basic and not necessarily understandable. ODS does not provide rich debugging features, and generally does not score well in terms of *security* features.

ODS does not provide features that support *data lineage* documentation and reporting. It also does not provide the ability to produce *impact analysis* reports. The level of *metadata* management is basic and allows only for mere editing and retrieval of few metadata attributes. This was clear during the evaluation process as it dealt with different data sources and pipelines and required a reasonable level of *data lineage* reporting. The low performance of ODS in terms of *traceability* and *auditability* is due mainly to absence or insufficiency of *metadata* management and *data lineage* reporting features. As *data transformation* capabilities of ODS require writing scripts, many changes were made to the *data transformation* scripts throughout the evaluation process. This highlighted the need for increased *traceability*, as the lack of version control for the transformation scripts made testing and modification difficult and lengthy.

The evaluation of ODS in terms of *performance* quality criteria showed that it can maintain high throughput and low response times under high load conditions. This enabled a high level of *adaptability* and *scalability* during the evaluation process, as the evaluation application required fetching and processing varying volumes of data at varying throughput requirements, and ODS could support these activities with a high level of *time efficiency* and relatively low level of *resource utilization*. In terms of *flexibility*, the evaluation process revealed that ODS workflows are constrained by the lack of support for wider range of data sources, transformation scripting languages, workflow management interfaces, integrations with third party tools, and integration with different *data warehousing* solutions. This also affects DF26, DF27, DF28, and DF30. Regarding *reusability*, the evaluation application required reuse of transformation scripts across some of the *data transformation* pipelines, and ODS could provide that functionality as its transformation scripting workflow allows for retrieval, reuse, and sharing of transformation scripts. This also applies for DF24. A reasonable degree of *extensibility* was achieved during the evaluation process, as the *Python* API client for ODS allowed for incorporating different API components to perform functionality not otherwise provided by standard API endpoints. *Smart execution* features could not be completely assessed as there exists only one feature under that category, which is periodic execution intervals. Other *smart* or conditional execution features are not supported in this version of ODS.

The evaluation application operated multiple pipelines consuming from multiple different data sources. Monitoring and managing such complex workflows requires a high level of *usability*. Within the scope of available features, ODS could provide a fair level of *usability*. ODS API documentation and available endpoints could cover the evaluation application requirements for programmatic configuration and management of data sources and pipelines. We could also test initiating and operating the same workflow through the ODS GUI, and we were able to fulfill the requirement of the evaluation application. ODS GUI allows for live-testing of transformation scripts. However, it does not provide clear warning messages or reports when the script contains errors. The GUI also allows for raw view or *configuration preview* of the data during the configuration of the data source. The GUI and the API of ODS were easy to use, within the scope of the supported features, and their documentation provided examples on the usage of both interfaces, which provided a high level of *understandability* and *ease of use*, which in turn enhanced *usability*. The deployment of ODS was easy and fast using the *Python* API client, which satisfies the criteria in DF39.

Regarding *cost efficiency*, ODS ETL component is is freely available with an *open source* license. It also does not require special system requirements for deployment. ODS deployment for the evaluation application was straightforward and did not require costs for technical support. This allowed for nearly no *cost of ownership* (DF41) and a high level of *costs efficiency*. Regarding the possibility of *no code* or *low code* operation and management, the evaluation application could test ODS support for this feature, and it was possible to operate and manage workflows through a GUI. However, ODS does not satisfy the criteria in DF33, DF34, DF37, and DF38, as those features are not supported in the current version of ODS. The language used for wiring transformation scripts in ODS is JavaScript. Through the evaluation process, many transformation scripts had to be written. This enabled testing the flexibility and richness of JavaScript as transformation scripting language. While it could accommodate the required transformations, it was inflexible and unsuitable at some points. This was highlighted by the contrast to the ease of performing data manipulation within the evaluation application code, as *Python* and *pandas* were used. Although ODS has a growing community of users, it cannot be considered large yet. This can be evaluated through the number of collaborators and contributors to ODS *GitHub* repository and *Slack* channel. However, the community is active and responsive, and could provide timely support that was needed at some stages of the development of the evaluation application, which satisfies DF42. Throughout the implementation of the evaluation application, clarifications about the usage and functionality of ODS components were needed. This required using ODS documentation frequently, which revealed that some sections of the documentation are either outdated, inconsistent, or provide incomplete coverage of the addressed features. Regarding *manageability*, workflows of the evaluation application could easily be monitored and managed through the GUI and the API of ODS. The ODS ETL components are provided under an *open source* license, which allows for greater *testability*, and satisfies DF45. In addition, some live testing features are provided through the transformations scripting interface.

5.2 Recommendations

Implementing the evaluation application helped assess ODS performance with respect to the most important quality criteria for an ETL to support *data science* activities. The evaluation process revealed shortcomings of ODS in practice, and highlighted pain points that the user encounters while deploying ODS in *data science* pipelines. This resulted in a set of recommendations for improvements that the ODS can implement in order to be more fit for use in *data science* contexts. For the sake of clarity, the recommendations are explained below in a list form.

R01 ODS should add *change data capture* capabilities.

R02 ODS should add *incremental update* capabilities.

R03 ODS should add *data slicing* capabilities

R04 ODS should allow users to execute *SQL* queries

R05 ODS should add *data profiling* capabilities.

R06 ODS should add *data lineage* documentation and reporting.

R07 ODS should add impact analysis reporting.

R08 ODS should provide better and easier to use *data cleansing* capabilities with less scripting.

R09 ODS should include a dedicated *schema mapping* interface with features that ensure ease of use and scalability.

R10 ODS should provide better and more extensive data documentation capabilities to improve interpretability.

R11 ODS should add entity recognition and matching capabilities.

R12 ODS should add *data enrichment* capabilities.

R13 ODS should support *OLAP* -based techniques (*OLAP cubes*, *slicing and dicing*, etc.).

R14 ODS should support *data streaming*.

R15 ODS should separate *data warehousing* and ETL functionality.

R16 ODS should allow integrations to external or third-party *data warehousing* solutions.

R17 ODS should improve OS-independence.

R18 ODS should add user authentication capabilities.

R19 ODS should allow a variety of user roles and access privileges.

R20 ODS should provide an interface for transformation and cleansing procedures sharing between users.

R21 ODS should provide an integrated library of reusable transformation and cleansing scripts and snippets.

R22 ODS should add *version control* capabilities for transformation scripts.

R23 ODS should allow *no code* or *low code data transformation*.

R24 ODS should allow *no code* or *low code schema mapping*.

R25 ODS should allow *no code* or *low code* mapping of workflow components.

R26 ODS should allow flexible choices for transformation scripting language.

R27 ODS should provide more extensive and advanced metadata management.

R28 ODS should improve traceability and reporting of ETL workflows execution.

R29 ODS should improve *flexibility* by supporting more protocols, formats, data sources, and integrations.

R30 ODS should provide integrations and connectors for widely adopted ETL and date warehousing tools and suites.

R31 ODS should support smart and conditional execution

R32 ODS should provide better error and diagnostics reporting.

R33 ODS should improve *recoverability*.

R34 ODS should seek wider adoption.

R35 ODS should improve community participation and engagement.

R36 ODS should allow forming custom notification messages.

R37 ODS should provide notification messages containing information that lead directly to the target data.

R38 ODS should allow accessing data of deleted *data sources* and *pipelines*.

R39 ODS should allow querying the database for available data sets.

In addition to the recommendations listed above, there has been some pain points, from the perspective of a user, that were experienced during the implementation of the evaluation application using ODS. It happens frequently that the *data scientist* user would need to fetch available historical batches of a data source before starting to periodically fetch recurring new data. This require-ment can be accomplished through the current version of ODS, but requires complex workflows and a lot of redundant actions by the user. Regarding the *notification* service, ODS can be set up to provide notifications of readiness of new or processed data from a pipeline. The notification message body contains very few information that is insufficient for achieving the implicit purpose of the notification, which is fetching the new or processed data. The notification mes-sage only provides the *pipeline* ID. This requires additional steps to fetch the data that the notification refers to. The user has to fetch the latest output of the *pipeline* using the *pipeline* ID through the *query* service, and hope that the latest *pipeline* output is the one that the notification refers to. Providing an *import* ID or a static link to the *query* API endpoint with the parameters leading to the target data would make the process easier and more accurate, as the user then would directly fetch the correct target data. Furthermore, it occurs sometimes that a *data source* or a *pipeline* is deleted, while the corresponding data, that has been already imported or processed, is still needed. Therefore, ODS needs to allow access to data of deleted *data sources* and *pipelines*. In an application that consumes many data sources, some times a new data analysis task emerges and there might be no need to fetch new data sets. Thus, it becomes necessary at some point to check the *query* service directly for available data sets in order to decide if a new data import is necessary or not. It would also be more useful and expressive if a five-line excerpt of the data was sent within the response. This five-line excerpt is very useful in deciding if a certain data set is suitable for the task or not, a similar feature exists in *pandas* data processing library using the *head* method from the *dataframe* class.

In order to provide more actionable conclusions, the recommendations were categorized and classified as follows:

- Based on desirability, ranging between:
  - very desirable
  - less desirable
- Based on absence from current version of ODS, ranging between:
  - absent
  - exists, but in need of improvement
- Based on association with one of the two role categories below:
  - *data science*
  - *data engineering*

- Based on importance, into three levels:
  - high
  - medium
  - low

Figure 5.1 visualizes the allocation of the recommendations into the aforementioned categorization. The graph follows the style of *Eisenhower matrix* to assess the importance of each recommendation. Recommendations that are both absent from ODS and very desirable are considered of high importance. Recommendations that are either absent and less desirable, or existent and very desirable are considered of medium importance. Lastly, recommendations that are both existent and less desirable are considered of low importance. Additionally, a color code was used to associate an additional layer of classification to the recommendations: red for *data engineering* related recommendations; and black for *data science* related recommendations.

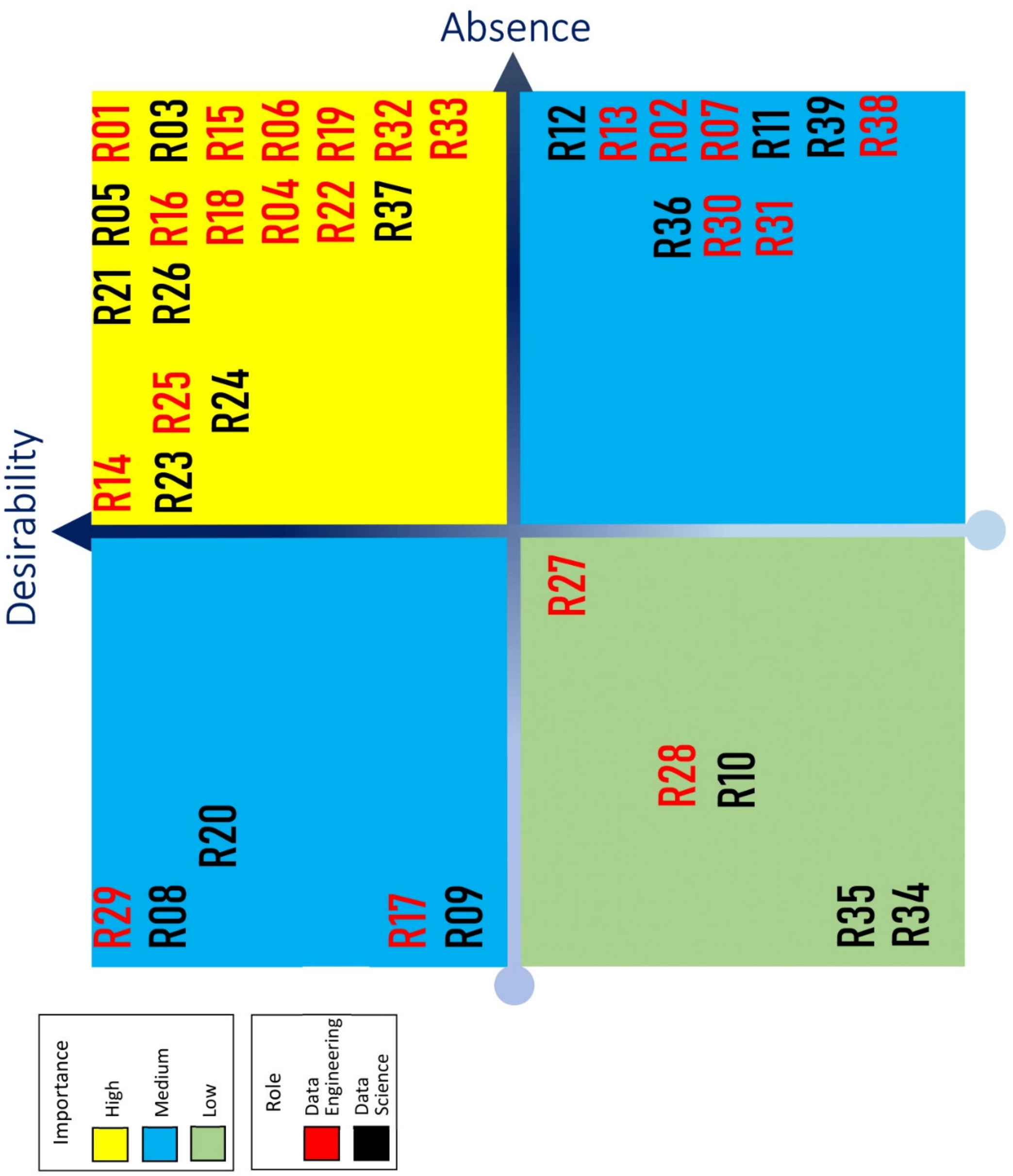

Figure 5.1: Allocation of concluded recommendations under different categorizes.

6 Conclusion

open data usability obstacles are mainly caused by inactive maintenance of data publishing initiatives. This causes large overhead on data usage activities downstream. Most of the effort in a *data science* project goes into overcoming data usability obstacles. To achieve stable and consistent feed of data, *data science* projects employ ETL tools to resolve *data quality* and *usability* issues. ODS was developed with a vision "to make consumption of *open data* easy, reliable and safe" through "decoupling of consumers from curators from publishers" so that collaborative innovation on using *open data* and fixing its quality issues becomes easier and faster (Riehle, 2019a). The main functionality of ODS is providing ETL processes, in addition to *data warehousing*. ODS has gone through multiple development cycles in order to get closer to its declared goals. In this study, we evaluated ODS v2 performance as an ETL in a *data science* context. The evaluation process was carried out using an evaluation application developed solely for the purpose of the study. The application could use most of the functionality provided in ODS v2. The evaluation process revealed some shortcomings of ODS that can be overcome with adding new features; improving existing features; or getting rid of some of the current features. A list of recommendations was put together in order to provide a road map for enhancing ODS *fitness for purpose* as an ETL for *data science*. The recommendations were then allocated under different classifications and categorizations in order to allow more actionable presentation. The findings concluded through this study have shed light on the strength points, such as *performance*, and the weaknesses of ODS, such as *inflexibility* and lack of *integrations*. The resulting recommendations, if implemented, can lead to wider adoption of ODS among the data science community, greater *usability*, and better *performance* in *data science* contexts.

Appendices

A *Data Science* Methodologies

A lot of methodologies and processes were developed to organize and execute data science projects. Some of these methodologies and processes date back to the early days of data science, when it was mainly referred to with the term *data mining*. Some of these earlier methodologies are still widely popular and influencing the recently developed methodologies, namely; Knowledge Discovery in Databases (KDD) and CRISP-DM. Fig 1 shows the effect KDD and CRISP-DM had on the evolution of later methodologies. Although CRISP-DM is heavily influenced by KDD, most later methodologies chose to build upon one of them or the other separately. In fact, the diagram shows KDD as an initial and CRISP-DM as a central approach for the development and evolution of later processes. As a concise and practical formulation of KDD, CRISP-DM prevailed to become the *de facto* standard for data science process till the moment (Martinez-Plumed et al., 2021).

KDD was introduced as a general framework for knowledge discovery that addresses all sub-processes needed for that purpose, from data preparation to model deployment. The authors of KDD made this shift in perspective clear by stating that "the distinction between the KDD process and the data-mining step (within the process) is a central point" (Fayyad et al., 1996, p. 3). Fayyad et al. (1996, 4:5) defined KDD process as "the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data". There are some underlying definitions to some of the terms mentioned in that brief declaration, which are necessary to understand the way KDD organizes the data science process and outcome. With adaptations from (Hamilton, 2000) and the original (Fayyad et al., 1996), explanations of those underlying terms are listed below:

- *Data*: a set of facts, *F*.
- *Model Representation*: a language *L* for describing discovered patterns.
- *Pattern*: An expression *E* in a language *L* describing facts in a subset F_E of *F*.
- *Non-trivial*: involves some search or inference; not a straightforward computation of predefined quantities.
- *Process*: KDD operations comprising many steps, all repeated in multiple iterations for refinement.
- *Valid*: true on new data with some degree of certainty.

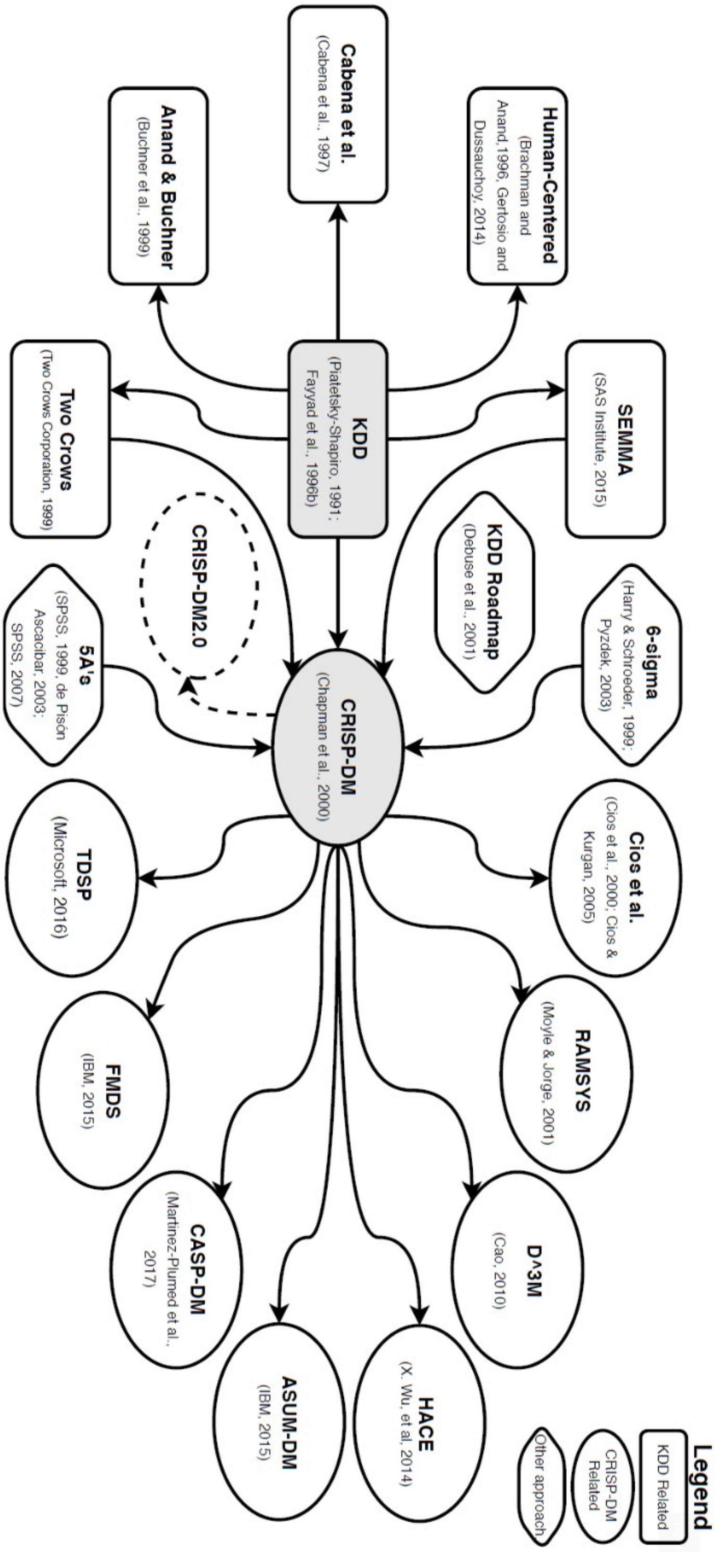

Figure 1: Evolution of most relevant Data Science models and methodologies (Saltz, 2020)

- *Novel*: not previously known to the system, and preferably to the user.
- *Useful*: actionable; leading to useful actions or benefit to the user or task.
- *Understandable*: leading to human insight, if not immediately then after some postprocessing.
- *Interestingness*: an overall measure of pattern value, combining validity, novelty, usefulness, and simplicity. "We can consider a pattern to be knowledge if it exceeds some interestingness threshold" (Fayyad et al., 1996, p. 5).

Fayyad et al. (1996) outlined the complete KDD life cycle in nine steps, with multiple iterations for refinement. KDD is an iterative and flexible process, which gives the user freedom to design a project's initial process and iterations in an agile manner. The basic flow of the KDD process is shown in figure 12. Using adaptations from (Mariscal et al., 2010), (Fayyad et al., 1996), and (Hamilton, 2000), the nine steps of KDD can be listed as follows:

- Learning the application domain, which includes:
  - Understanding the application domain
  - Learning relevant prior knowledge
  - Identifying goals of the process
- Creating a target data set on which discovers is to be performed, which includes:
  - Selecting a data set
  - Focusing on a subset of variables or data samples
- Data cleaning and preprocessing, which includes:
  - Removal of noise or outliers
  - Collecting necessary information to model or account for noise
  - Strategies for handling missing data fields
  - Accounting for time sequence information and known changes
  - Dealing with data management challenges
- Data reduction and projection, which includes:
  - Goal-oriented feature engineering of the target data
  - Applying dimensionality reduction techniques
  - Executing data transformations

  - Finding invariant representations of the data
  - Reducing effective variables
- Data mining task selection matching process goals (step 1) to a data mining taskFor example, regression, classification, dependency modeling, forecasting, and so on.
- Data mining algorithm selection, which includes:
  - Exploring algorithm(s) that can be used for the selected task
  - Selecting method(s) for searching for data patterns
  - Deciding which models and parameters are appropriate for the data
  - Matching a particular data mining method with the overall criteria of the KDD process
- Data mining
- Interpretation, which includes:
  - Interpreting discovered patterns
  - Reiteration over any of the previous steps if needed
  - Visualization of discovered patterns
  - Removing redundant or irrelevant patterns
  - Translating useful patterns into terms understandable by the users
- Acting on the discovered knowledge, which includes:
  - Consolidating discovered knowledge
  - Using the knowledge directly, or incorporating it into another system for further action, or simply reporting it to interested parties
  - Checking for and resolving potential conflicts with prior knowledge

It is worth noting that the KDD process may require significant iteration and "can contain loops between any two steps" (Fayyad et al., 1996, p. 6). A basic flow,like the one depicted in figure 2, may not reflect the flow in a real data science project, as many of the outlined steps may require multiple iterations to be fairly accomplished, and some steps may be irrelevant in some cases dependin on the quality of the acquired data (Kurgan & Musilek, 2006).

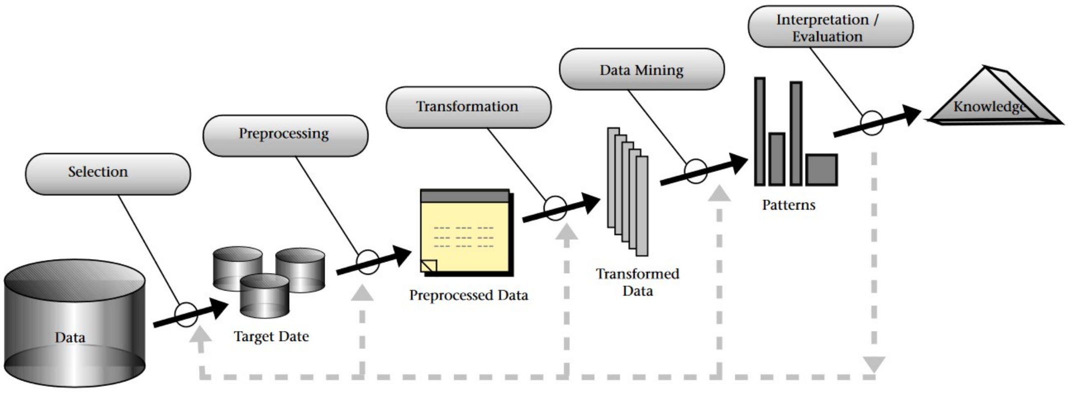

Figure 2: An Overview of the Steps That Compose the KDD Process (Fayyad et al., 1996)

KDD process is more complex in practice, and involves more elements than those modeled in the original process. For example, KDD process requires a lot of decision-making by the user throughout the process steps and iterations. Another approach, the *human-centered approach* was developed to take some of those elements into account (Gertosio & Dussauchoy, 2004). The *human-centered approach* addresses the interactive nature of the KDD process in practice, and emphasizes the role and the interactive involvement of the human element throughout the process. Fig 3 shows the process flow according to the *human-centered approach*. As the name indicates, the *human-centered approach* incorporates the role of the data analyst or miner and addresses tasks from the viewpoint of the human element, which has the advantage of highlighting the decisions that a user has to make.

Despite this shift of perspective, the *human-centered approach* did not stray from the KDD process principles. In fact, it was considered a completion of the KDD model (Gertosio & Dussauchoy, 2004). The *human-centered approach* consists of the following steps:

- Task discovery, which corresponds to the first step in the KDD process.
- Data discovery, which corresponds also to the first step in the KDD process.
- Data cleaning, which corresponds to the second, third, and fourth steps of the KDD process.
- Model development, which corresponds to the fifth, sixth, and seventh steps of the KDD process.
- Data analysis, which corresponds to the eighth step of the KDD process.

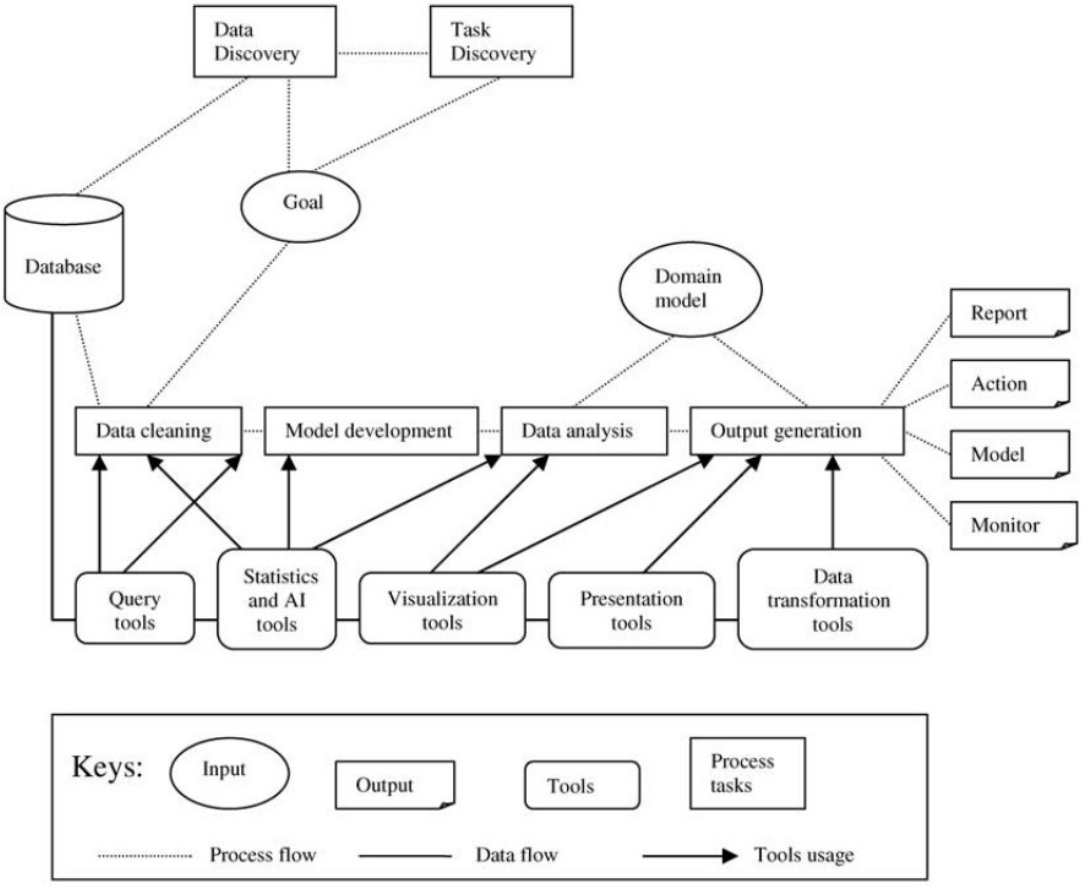

Figure 3: The *human-centered approach* (Gertosio & Dussauchoy, 2004)

- Output generation, which corresponds to the ninth step of the KDD process.

Another approach that is widely in-use is the Sample, Explore, Modify, Model, Assess (SEMMA) approach. It was developed by SAS Institute, which is one of the market leaders of statistics and business analytics software. The steps of data science process in the SEMMA approach are shown in Figure 4. SEMMA was created to organize the data mining process for SAS customers with greater focus on model development and less focus on preceding and succeeding operations, or as SAS Institute described it "a logical organisation of the functional tool set of SAS Enterprise Miner for carrying out the core tasks of data mining" (SAS Institute, 2012). SEMMA is facilitated by an integrated GUI within the Enterprise Miner software.

SEMMA approach assumes that the user has already learned the application domain, which ignores the first step in KDD approach. It also does not incorporate usage of discovered knowledge in the process model, in contrast to the ninth

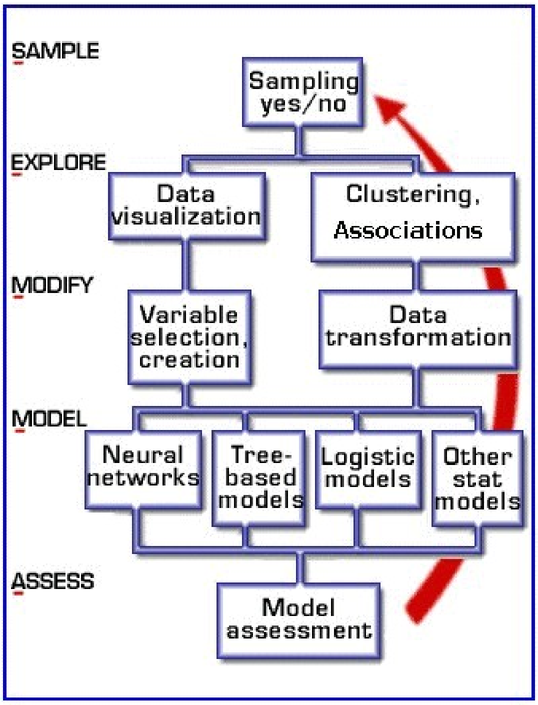

Figure 4: SAS Institute SEMMA approach (SAS Institute, 2017)

step in KDD approach. The steps of SEMMA approach, as summarized in (SAS Institute, 2017), can be listed as follows:

- Sample: create subsets of the data that are large enough to contain significant information, yet small enough for efficient processing.
- Explore: search the data for anticipated relationships, anomalies, and trends.
- Modify: apply transformations, feature engineering, and dimensionality reduction of the data for the sake of efficient modeling.
- Model: create a model using a data mining modeling technique.
- Assess: evaluate the usefulness and reliability of the findings.

Another important data science methodology is the *Two Crows* data mining process model. It was developed by the Two Crows Corporation in 1999 based on a previous edition of the same model, in addition to some insights from the very early version of CRISP-DM approach (Mariscal et al., 2010). The data science process, under the Two Crows model, does not follow a linear path. Despite being based on KDD approach, the Two Crows model addresses the practical need of looping back and forth between process steps more expressively than the KDD approach (Two Crows Corporation, 1999). An outline of the process steps and the possible loops in the Two Crows model is shown in figure 5.

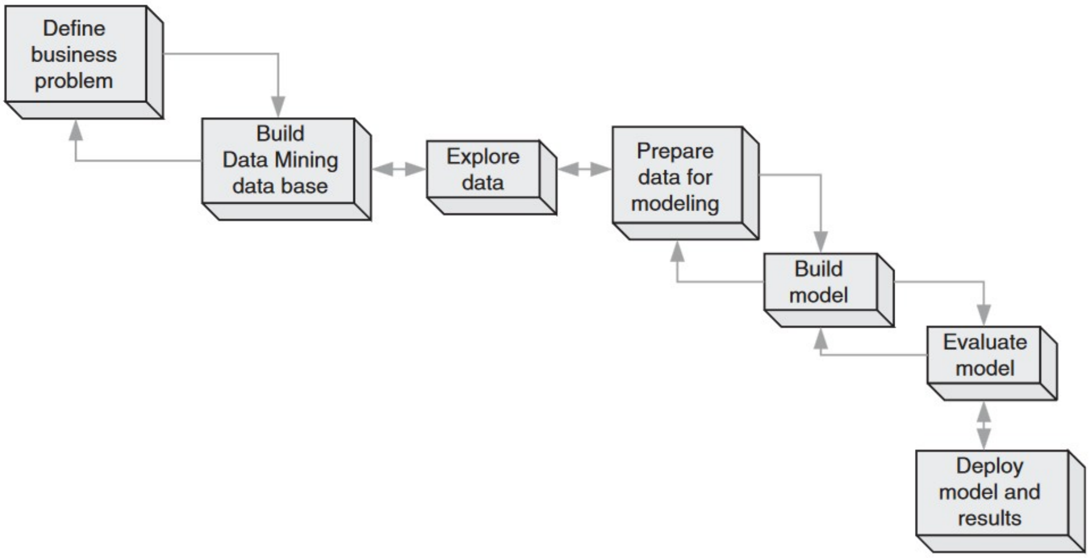

Figure 5: *Two Crows* data mining process model (Mariscal et al., 2010)

The most popular and widely adopted data science process model is CRISP-DM (Martinez-Plumed et al., 2021). A consortium of companies with interest and experience in data mining was created in order to study and improve the data mining process. The consortium included organizations such as; Teradata, SPSS -ISL-, Daimler-Chrysler, and OHRA. At a later stage, a boost, in the form of funding from the European Commission, helped the group aim higher and work on a mature standard process model for data mining that would be non-proprietary and freely available (Chapman et al., 2000). The composition of the consortium contributed to making CRISP-DM "industry-, tool-, and application-neutral" (Mariscal et al., 2010), which was a major reason for the wide adoption of the process model.

CRISP-DM addresses the life cycle of data mining projects by organizing in a hierarchical manner with vertical and horizontal relationships. It uses four levels of abstraction:

- Phase
- Generic task
- Specialized task
- Process instance The four level breakdown of CRISP-DM methodology is shown in figure 6.

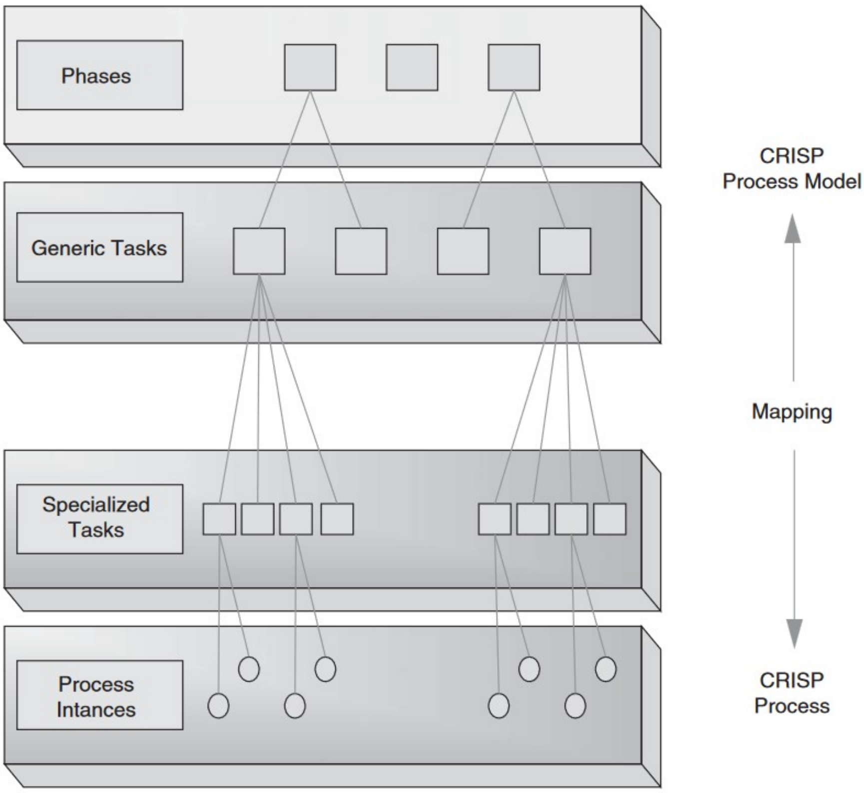

Figure 6: Four-level hierarchical breakdown of CRISP-DM process (Chapman et al., 2000)

At the top level of the hierarchy, phases, CRISP-DM organizes the life cycle of a data science project into six phases, as shown in figure 7. The particular order depicted by the arrows in figure 7 indicates the most important and frequent path across phases. This, and the hierarchical organization embedded in CRISP-DM, may indicate that the data science process in CRISP-DM follows a waterfall life cycle. However, the CRISP-DM method does state that: 'The sequence of the phases is not rigid. Moving back and forth between different phases is always required. It depends on the outcome of each phase which phase or which particular task of a phase, has to be performed' (Chapman et al., 2000, p. 13).

The CRISP-DM modeling of the data science process goes deeper into details that help organize practical aspects of data science projects. In fact, the level of detailed guidance is remarkable. The CRISP-DM user guide presents a model for sub-processes and activities needed to accomplish the aforementioned phases and the underlying tasks. It proposes specific activities to produce each output (Chapman et al., 2000, 35:68). This comprehensive modeling might be a main reason behind the wide acceptance and adoption of CRISP-DM.

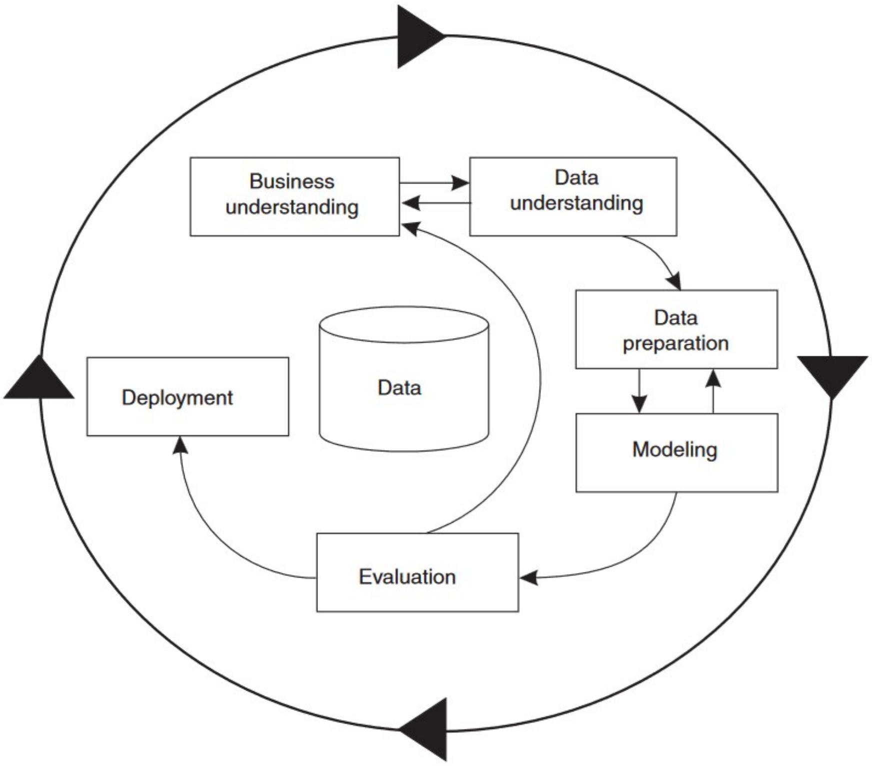

Figure 7: The six phases of a data science project as proposed by CRISP-DM methodology (Chapman et al., 2000)

As shown in figure 1, a lot of data science methodologies have been developed based on CRISP-DM. However, CRISP-DM v1.0 is still the most widely used and adopted methodology. In 2006, an Special Interest Group (SIG) was formed and announced a project to upgrade the methodology and create CRISP-DM v2.0, but the project stalled at some point, and it is unknown how much progress was made towards that goal (Martinez-Plumed et al., 2021). Many methodologies were developed to build upon CRISP-DM v1.0 and address its shortcomings, but most of them failed to gain traction or adoption.

RAMSYS is another important data science process model that became popular because of the way it organizes distributed and distant collaboration. RAMSYS addresses missing aspects in the CRISP-DM model, and can be considered a refinement of it (Martinez et al., 2021). RAMSYS supports collaboration of distributed teams on data science projects while allowing for effective management of the steps and the outcome of the process, and ensuring an orderly flow of information between parties. It tries to organize the data science process so that problem solving, knowledge sharing, and ease of collaboration are collectively achieved between geographically distant and distributed teams. RAMSYS classifies roles of data mining units or "nodes" in the "expertise network" in a data science project into three categories:

- Modellers
- Data masters
- Management committee

The steps of RAMSYS process model is similar to those in the CRISP-DM process, but with the addition of a new task, which is *Model Submission*, as illustrated in figure 8. This new task is in line with the constant communication and knowledge sharing required by the RAMSYS process as it gives modellers and data mining units freedom in creating their own models, but requires the models to conform to the agreed-upon evaluation scheme and to be shared and submitted to the *information vault* (Martinez et al., 2021).

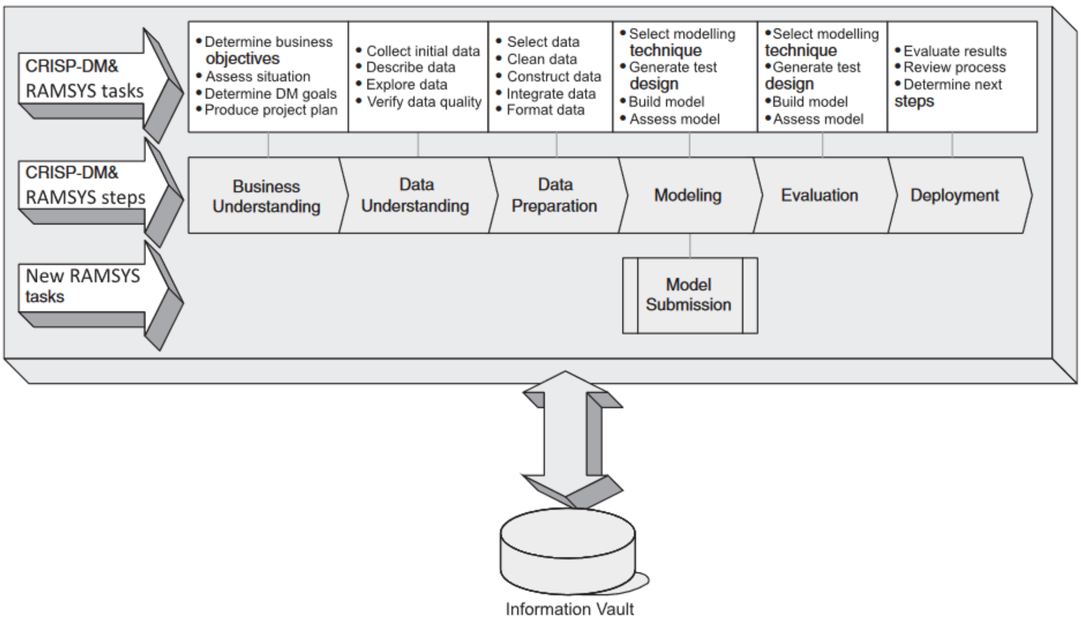

Figure 8: Steps of the RAMSYS methodolgy (Mariscal et al., 2010)

According to (Moyle & Jorge, 2001), the RAMSYS methodology adhere by a set of high-level guiding principles, which are designed to accommodate distributed work groups in a way that seemed futuristic at the time. Adapted from (Moyle & Jorge, 2001), those principles are:

- Light management
  - The problem and the objectives should be clear from the beginning to all participants

  - The *management committee* role is not to micro-manage each node or unit

- Start any time
  - *Problem information* to start problem solving should be available all the time to ensure smooth particpation of new expertise if needed
  - Project participants to push tasks outputs to the *information vault*

- Stop any time
  - Problem solving should be conducted in a way that ensures a working solution is available whenever the *management committee* issues a stop signal
  - Simpler models are tried first

- Problem solving freedom
  - Each team in the network can choose their approach to solve the problem
  - The *management committee* may give suggestions, but does not prescribe problem-solving approaches

- Knowledge sharing
  - Each modeller can produce new knowledge, and it should be shared immediately with the rest of the network

- Security
  - Project data is not to be shared outside the project
  - The *management committee* must control and monitor access to project information

- Better solutions
  - As each node is free to follow its own approach, a range of solutions are produced
  - The combination of solutions may form a better solution

The RAMSYS methodology proposed many novel and interesting concepts (Martinez et al.,2021). A concept that is relevant to our study is the *Information Vault*.The information vault is an artifact that enables involved parties to standardize and streamline communications and knowledge sharing. According to (Moyle & Jorge, 2001), the information vault should contain:

- Problem definition

- Distilled knowledge from related problems
- Evaluation criteria definition
- Data
- Hypothesis investment account

As the nature of the items contained in the shared *information vault* indicates, RAMSYS methodology stresses constant, real-time, and frequent communication according to a predefined standard at each step. Martinez et al. (2021) conducted a review of nineteen of the most popular data science methodologies to assess how well they address data science projects main challenges. The review assessed the methodologies against twenty-one challenges under three categories: team management, project management, and data and information management. The review revealed that RAMSYS methodology achieved the highest integrity score among all the reviewed methodologies. The results of the review are shown in figure 9.

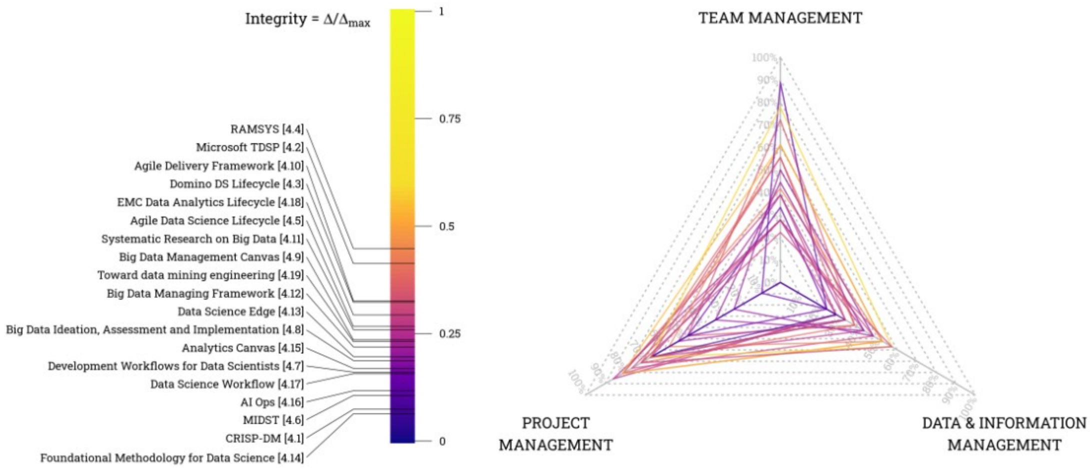

Figure 9: Quantitative summary of the reviewed methodologies: a) integrity value is represented on the bar plot and b) each category's scores are illustrated on the triangular plot, with the line color representing the integrity (Martinez et al., 2021)

Another popular methodology that stresses agility, collaboration, and knowledge sharing is TDSP from Microsoft Corporation. The process is presented as an "agile, iterative data science methodology that helps improving team collaboration and learning" (Microsoft, 2022). However, describing it as a methodology may be contested, as it relies heavily on the Microsoft ecosystem of products, which reduces its validity for use outside those systems (Martinez et al., 2021). An extensive documentation is available for the TDSP method, in addition to

Microsoft support with tools and utilities at every level of implementation. According to the documentation in (Microsoft, 2022), the process of TDSP entails four components:

- A data science life-cycle definition
- A standardized project structure
- Infrastructure and resources recommended for data science projects
- Tools and utilities recommended for project execution

The core part of the methodology is the definition of data science life cycle. An illustration of the TDSP life cycle is shown in figure 10. The overall structure inherits from both KDD and CRISP-DM. It defines the project life cycle using stages, tasks, and artifacts. Those tasks and artifacts are associated with the proposed set of project roles listed below:

- Solution architect
- Project manager
- Data engineer
- Data scientist
- Application developer
- Project lead

A grid view of the stages and roles along with the corresponding tasks and artifacts is shown in figure 11. The artifacts shown in the figure contribute to achieving the second component of TDSP: Standardized project structure. Another element for achieving the standardized project structure is a proposed set of directory structures, and templates for project documents, code, and models. At this point, the TDSP process starts to stray from the neutrality and tool-independence required for a valid methodology. This is mainly because the level of consistency, and connectivity that this component of the process requires might not be easily attainable outside Microsoft’s comprehensive ecosystem of integrated solutions. The TDSP recommends and promotes cloud solutions for storage and analytics to enhance collaboration and knowledge sharing. In that regards, it introduces concepts similar to those presented by the RAMSYS process model, for example, the *project charter* artifact (Martinez et al., 2021).

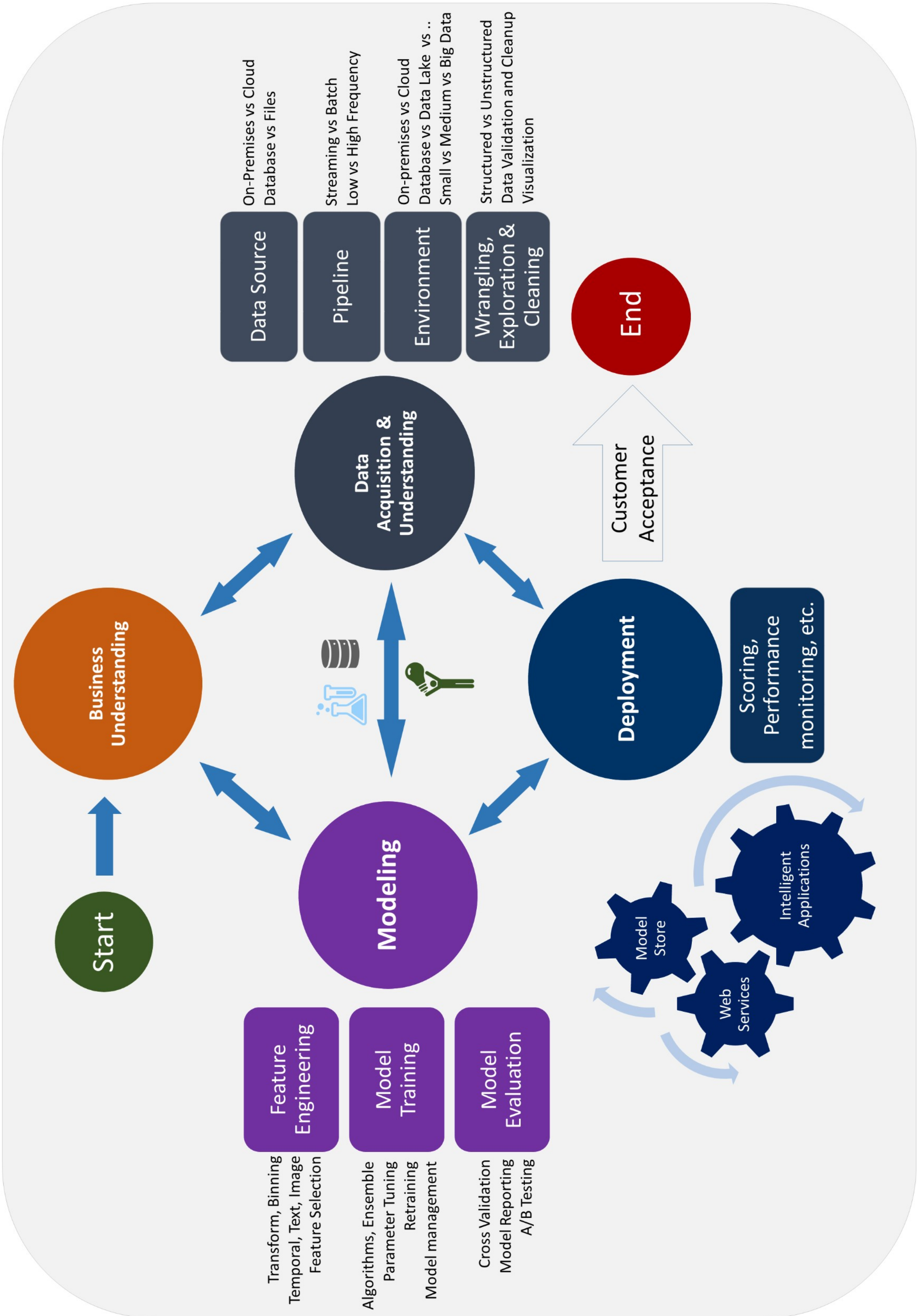

Figure 10: Visual representation of TDSP life cycle (Microsoft, 2022)

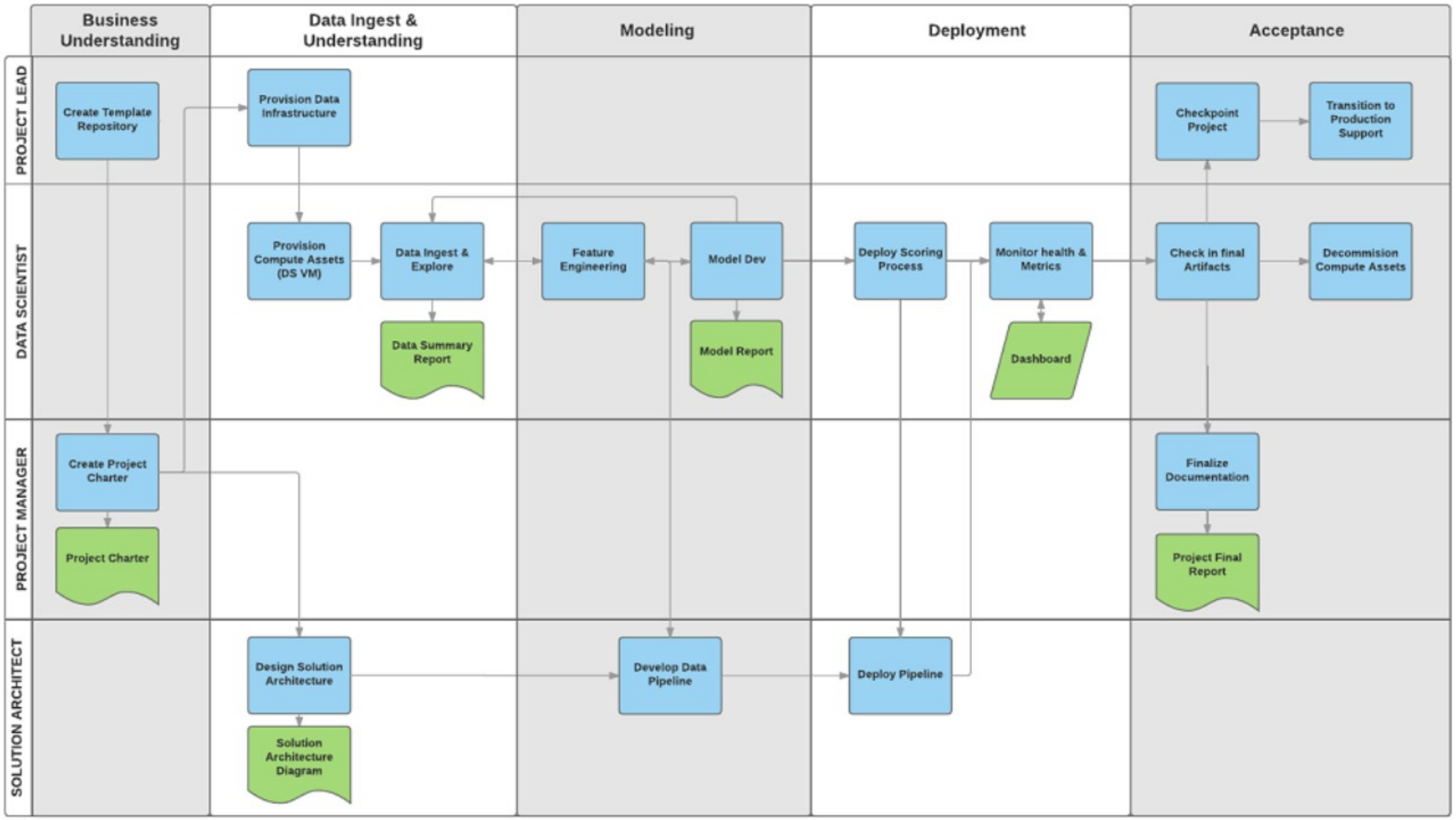

Figure 11: Tasks (in blue) and artifacts (in green) corresponding to stages (on the horizontal axis) and roles (on the vertical axis) in a data science project life cycle in the TDSP process model (Microsoft, 2022)

B ETL Desirable Qualities and Corresponding Metrics

ETL is at the core of data warehousing systems, and its processes are critical for the success of dependent data science activities. Vassiliadis et al. (2002) reported that ETL development occupies up to %80 of the development time and third of the efforts and expenses in a data warehouse project. Furthermore, ETL processes cost more than half of the total runtime costs of a data warehousing system. However, many companies prefer to build their in-house ETL solutions to cover all their process needs. This is in part due to the lack of research on ETL processes and methodologies, which makes solutions follow ad-hoc methodologies. Consequently, deployment of a new ETL solution becomes more complex and requires long training and steep learning curves. Vassiliadis et al. (2002) proposes a conceptual model for ETL processes that considers the mapping of attributes from source data system to target data system as the core deliverable from an ETL design process. The proposed conceptual model also enables custom inter-attribute relationships, extensibility to accommodate patterns for ETL activities, and reusability of frequently used ETL activities, especially by incorporating data cleansing activities into the model.

An extensive survey by The Data Warehousing Institute (TDWI) examined the hurdles and challenges that face developers while working with ETL solutions. The report discussed business requirements that are behind new features in ETL solutions. For the reasons discussed earlier, there is always a debate about whether to buy or build ETL solutions. The report also taps on that debate and discusses the pros and cons of following each approach. The report provides a unique perspective on ETL desirable qualities and evaluation criteria, as it was based on the interviews and responses from 1051 participants most of whom were industry experts who implemented ETL solutions, data scientists, and business analysts. The report also incorporates results from a previous survey of more than 1000 business intelligence professionals that TDWI conducted in 2002 (Wayne Eckerson & Colin White, 2003).

The report explores the most important pain points in the usage of ETL solutions from the perspective of expert users and developers. It mentions that experts prefer ETL tools that reduce the need for user-written procedures, as those increase complexity and maintenance cost. The report also revealed that an enhanced graphical development interface is a highly desirable feature as it makes an ETL easier to use. The report stresses that data volumes, sources, and granularity are ever-increasing, and this creates a need for ETL tools to improve reliability, capacity, and processing speed. The report also underscores the ever-increasing diversity in data sources that a data warehousing system deals with. This data source diversity is associated with type diversity as well, as the responses shown in figure 12 indicate.

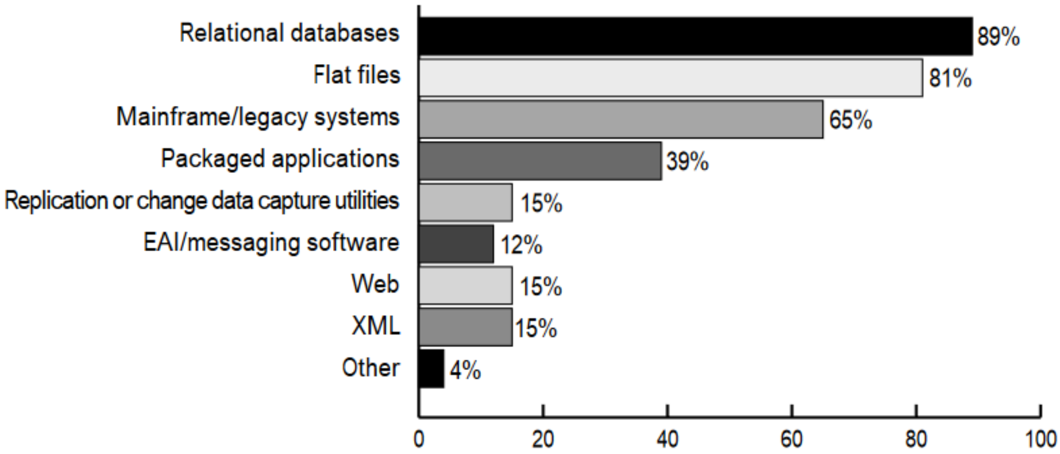

Figure 12: Types of data sources that ETL programs process. Multi-choice question, based on 755 respondents. (Wayne Eckerson & Colin White, 2003)

As the role of business analytics and data-dependent systems in decision-making and business operations is becoming increasingly important, high availability of the ETL component becomes essential for a functioning data warehousing system. The report also states that, as a result of data sources diversity, ETL systems need to support variable update cycles that match different data publishing schedules. The report also discussed the importance of data quality capabilities and other add-on components that are very desirable for data engineers and data scientists. As fig 13 shows, inclusion of data cleansing, profiling, and analytics capabilities within an ETL solution is highly desirable.

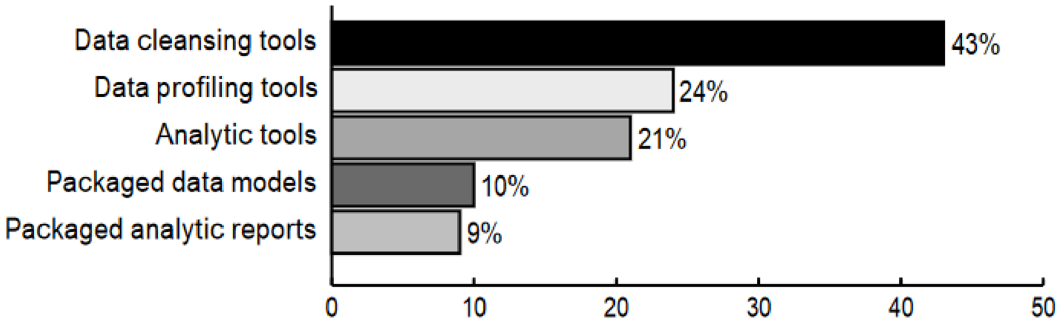

Figure 13: Desirability of certain add-on components, based on 740 respondents (Wayne Eckerson & Colin White, 2003).

Responses in the survey although showed that users want ETL solutions to support global meta data management. This means automatic documentation, coordination, and management of metadata corresponding to the various data sources and sets within data modeling tools, sources, data marts, data warehouses, data analytics components, portals, and repositories, and the interdependencies among them and among their elements (Wayne Eckerson & Colin White, 2003). However, some users doubt that an ETL solution can achieve these requirements. A global metadata management approach would greatly enhance data consistency and standardization, especially across large networks of connected analytics and data warehousing solutions. Another aspect that has been discussed through the report, is the seamless fitting of the ETL to different components in the data infrastructure of a business environment. This requires ETL solutions to support data integration processes, not only from external sources, but also internally across the board. The survey results also showed that ease of deployment is a critical parameter in evaluating an ETL solution. As shown in figure 14, it ranked first among the reasons that can motivate a purchase decision of an ETL solution. The figure also shows how important data integration and global metadata management capabilities are to data scientists and engineers, as these features ranked second and third in the surveyees responses about reasons that would make them favor buying an ETL solution. That said, the debate of building versus buying ETL solutions is not totally settled. In some business environments, it might be infeasible or inefficient to buy an ETL solutions. We may expect open data consumers to be represented more in that group. That debate was also discussed in the report, including responses of users explaining the pros and cons of each approach. Figure 15 shows the ranking of the reasons that may make experts rule in favor of building an ETL tool instead of buying it. That debate, articulated by the responses shown in figures 14 and 15, highlights the importance of *cost of ownership* as a key criteria in evaluating ETL tools. This is confirmed even further by the responses from survey participants to the question: "How Does Pricing Affect Your ETL Purchasing Decision?", shown in figure 16. Seventy-one percent of participants consider pricing among the three most decisive factors for purchasing an ETL solution. Ninety-six percent of participants consider pricing generally important.

Deploying an ETL tool into a data warehousing system or a data integration pipeline entails many challenges. According to responses in (Wayne Eckerson & Colin White, 2003), ETL deployment can be less challenging if the tool has data profiling and cleansing capabilities that are reliable enough to ensure adequate data quality and seamless integration of the data sources from which the project or the data warehouse draw. In response to that part of the survey, respondents described many challenges that complicate the deployment of ETL tools into their data ecosystem. We have mentioned some of these pain points earlier, but

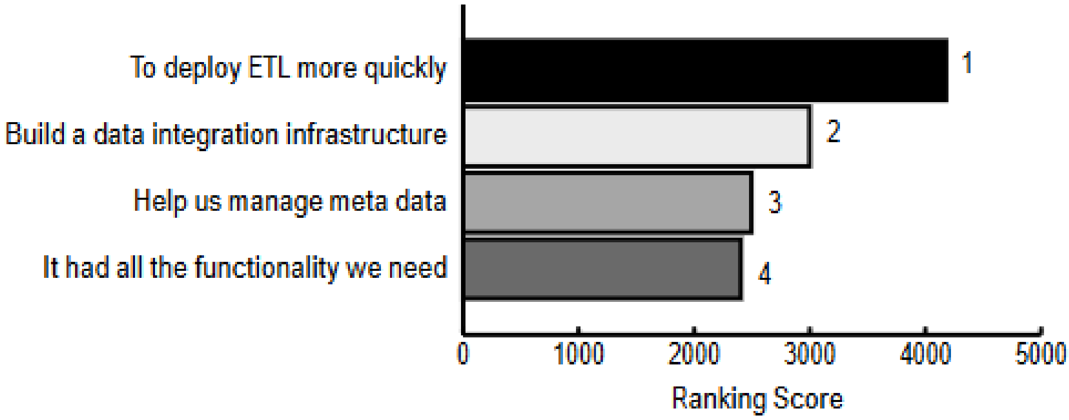

Figure 14: Ranking of possible motives for a purchase decision an ETL solution (Wayne Eckerson & Colin White, 2003).

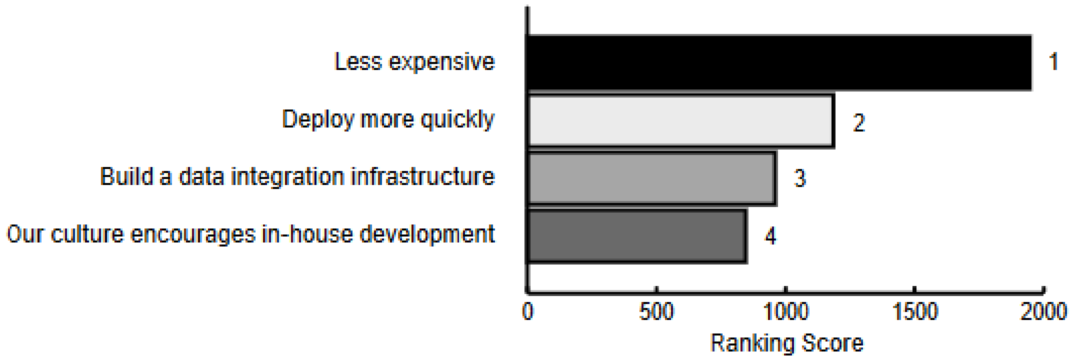

Figure 15: Ranking of possible motives behind favoring building an ETL tool instead of buying it (Wayne Eckerson & Colin White, 2003).

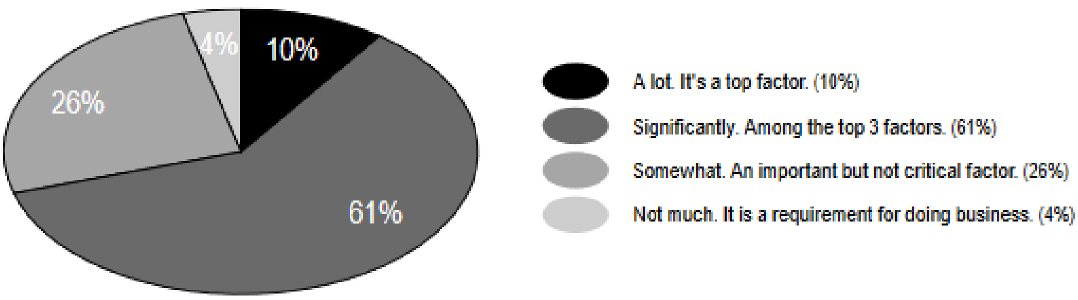

Figure 16: Importance of pricing as a factor in a purchase decision for an ETL solution (Wayne Eckerson & Colin White, 2003).

some of the challenges described in that part are yet to be mentioned. Figure 17 shows ranking of the pain points in dealing with ETL solutions, with the most painful at the top. In addition to the challenges and the desirable qualities mentioned before, it shows that users highly value both the ease of use and the ease of learning of the ETL tool. It also shows that scalability is a highly desirable quality that reduces complexity of the process. Integration with third-party tools and applications was ranked among the ten most challenging ETL-related tasks. The users also stressed the extensibility of the ETL tool as they complained about the complexity of adding and "integrating user-defined functions" into the ETL tool. Challenges such as extensibility, ease of use, ease of learning, and ease of finding skilled ETL developers can be all mitigated if the ETL tool source code is open and well-documented. Another factor that may help mitigate these challenges is the language used to write transformations and schema mappings inside the ETL tool, which has to be easy to learn and use. Many programming languages can be fit for the purpose of writing complex transformations, but they vary in the ease of learning and usage. This issue was discusses in the 2021 *State of Data Science* survey conducted by *Anaconda*. The responses visualized in figure 18 show clearly that some programming languages are significantly more usable for the data science community than others, which indicates that those languages are better equipped for implementing data science tasks including data cleansing and transformation.

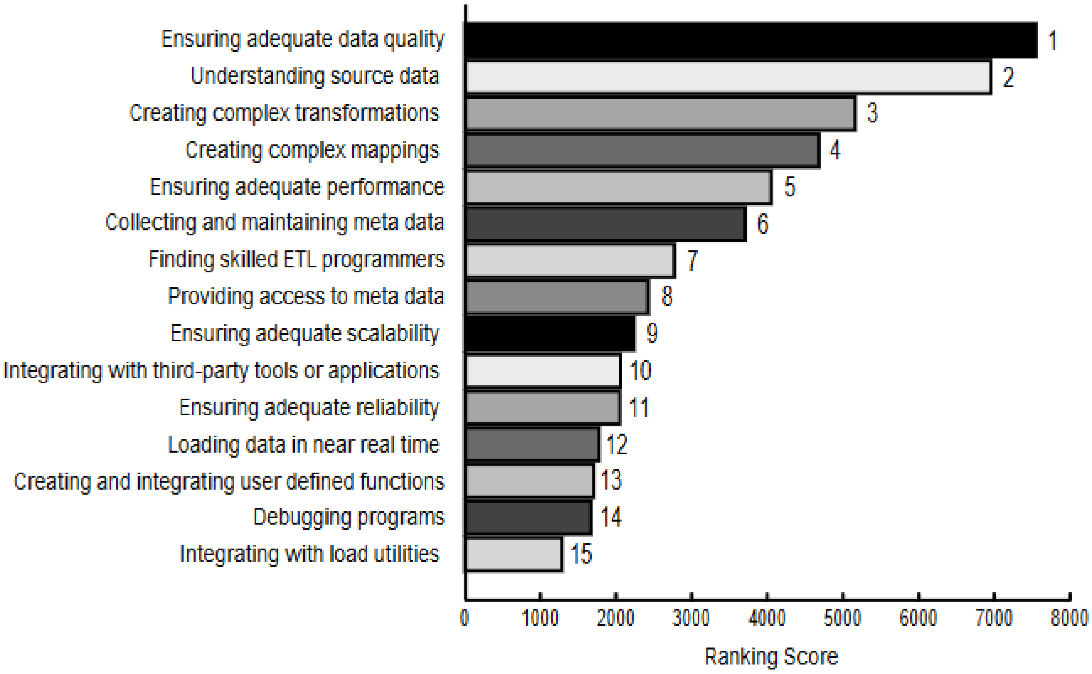

Figure 17: Most challenging ETL-related tasks (Wayne Eckerson & Colin White, 2003).

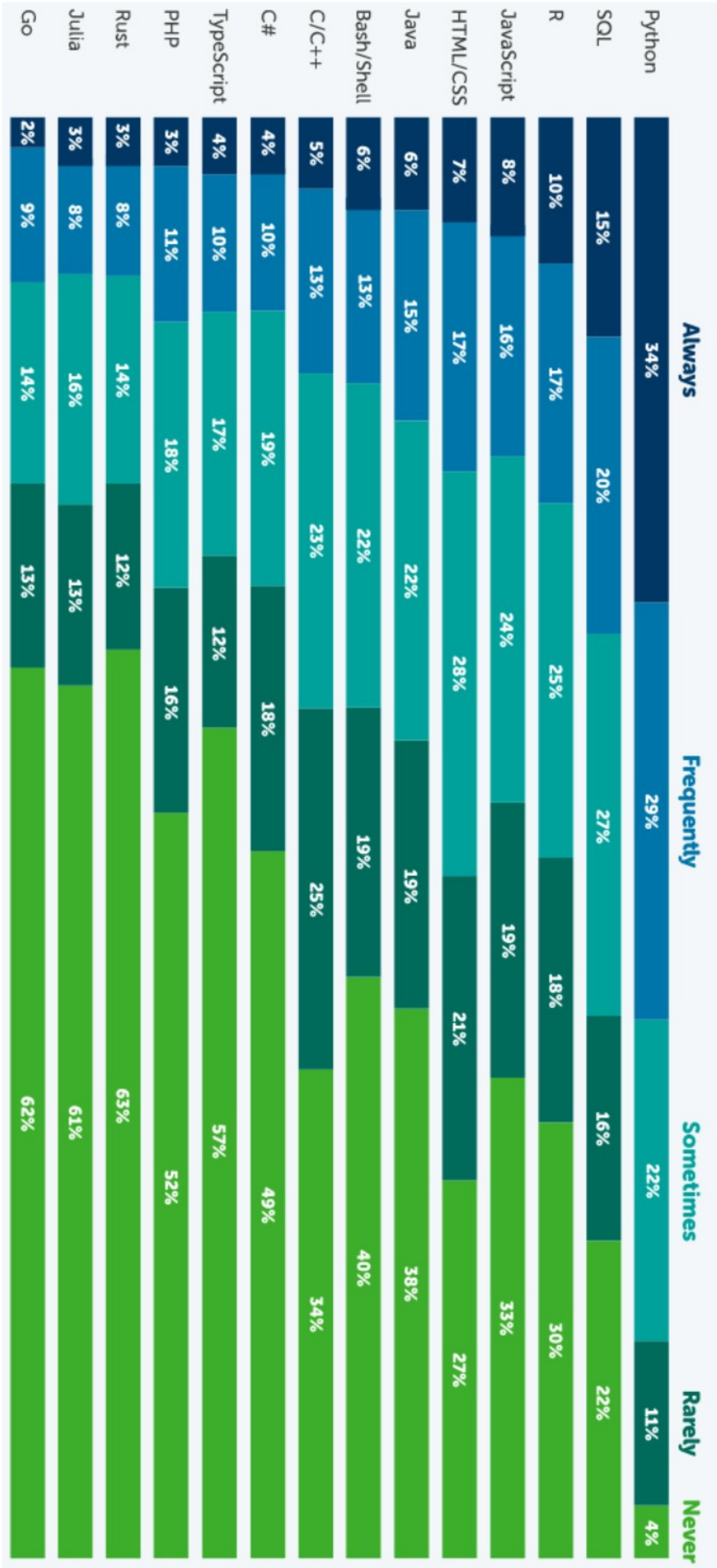

Figure 18: Responses to the question "How often do you use the following languages?" from 3104 survey participants (Anaconda Inc., 2021)

For an ETL tool to support a vibrant data pipeline, it needs to ensure reliable and efficient job execution. Those job execution qualities can be further enhanced by allowing for "smart" execution. For example, conditional execution based on content of the data batch, predefined thresholds, or inter-process dependencies (Wayne Eckerson & Colin White, 2003). In addition, it is highly desirable for an ETL to have robust debugging and error recovery capabilities, which "minimize how much code developers have to write to recover from errors in the design or runtime environments" (Wayne Eckerson & Colin White, 2003, p. 25). Error reports and diagnostics need to be clear, understandable, and actionable. "Instead of logging what happened and where, users want the tools to say why it happened and recommend fixes" (Wayne Eckerson & Colin White, 2003, p. 25). It is far more efficient and desirable to incrementally update the data warehouse instead of rebuilding from scratch every load. This feature requires the ETL tool to have *change data capture* capabilities, which is a highly desirable feature as per the survey respondents in (Wayne Eckerson & Colin White, 2003). *Change data capture* capabilities allow the ETL tool to fetch only the changes that have occurred after the last load. It requires a combination of *Change Capture Agents*, *Change Data Services*, and *Change Delivery Mechanisms* in order to execute successful batch-oriented (*pullCDC*) or live CDC (*pullCDC*) (Ankorion, 2005). Figure 19 illustrates the importance of the aforementioned features from ETL users perspective.

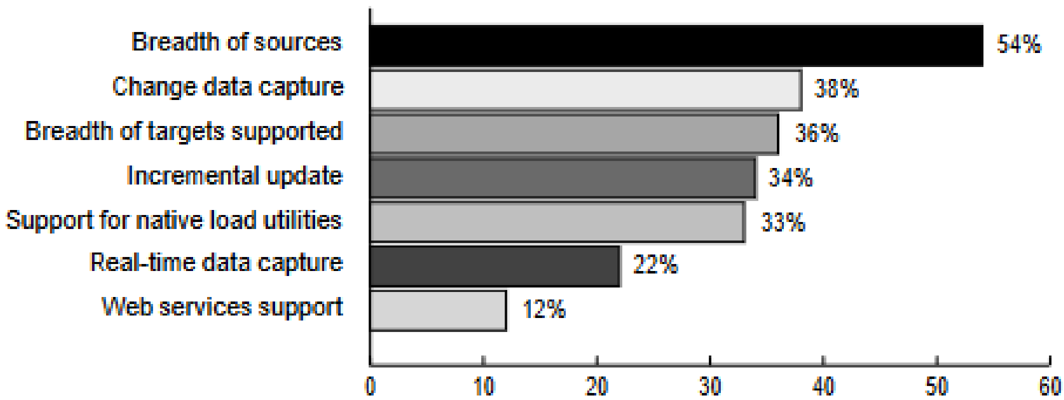

Figure 19: Percentages of survey users who marked the above ETL features as "very important". Percentages are based on responses from 745 participants (Wayne Eckerson & Colin White, 2003).

The comprehensive survey in (Wayne Eckerson & Colin White, 2003) grouped features that can be examined to evaluate an ETL tool into the following categories:

- Design features

- Meta data management features
- Transformation features
- Data quality features
- Performance features
- Extract and capture features
- Load and update features
- Operation and administration features

In the part of the survey that addressed *design features*, survey respondents showed immense interest in features that enhance *ease of use*. Seventy percent of survey respondents marked *ease of use* as "very important". Results showed that a graphical development environment is a highly desirable feature that is critical for *ease of use* and *ease of learning*, as 84 percent of survey respondents rated a "visual mapping interface" as either "very important" or "fairly important" (Wayne Eckerson & Colin White, 2003). Responses also showed that reusability of objects, tasks, and processes is very important for *ease of use*. Survey results also revealed that *ETL* users consider the choice of transformation language and the power of transformation capabilities as very important and critical to the quality of an ETL tool. Responses also showed that strong debugging capabilities are considered very important by a majority of ETL users and developers. Remarkably, a big portion of survey participants rated *openness* as a very important design feature, which is a reasonable choice as *openness* would enhance most of other desirable features. Figure 20 shows the rankings of design, transformation, and meta data features that were voted as "most important" the most. It is clear from the ratings in the figure that *meta data management features* may be less critical than design and transformation features. However, being marked by nearly 40 percent of survey participants as "very important" indicates that is is a very desirable set of features for an ETL tool. As shown in the figure, users showed interest in having interfaces for meta data visualization, querying, and management. Meta data reports, such as *impact analysis* and *data lineage* reports, proved to be of high importance to ETL users and developers according to responses in the *meta data management features* section of the survey.

Performance features are generally desirable in almost any software product. Thus, the vast majority of respondents to the survey in (Wayne Eckerson & Colin White, 2003) rated performance features as "very important". Eighty-six percent of survey participants ranked *reliability* as a "very important" performance features, which is the highest rating of any feature throughout the survey. Survey participants highly rated the importance of other performance features, such as *throughput*, *scalability*, and *availability*. Figure 21 shows the *performance features*

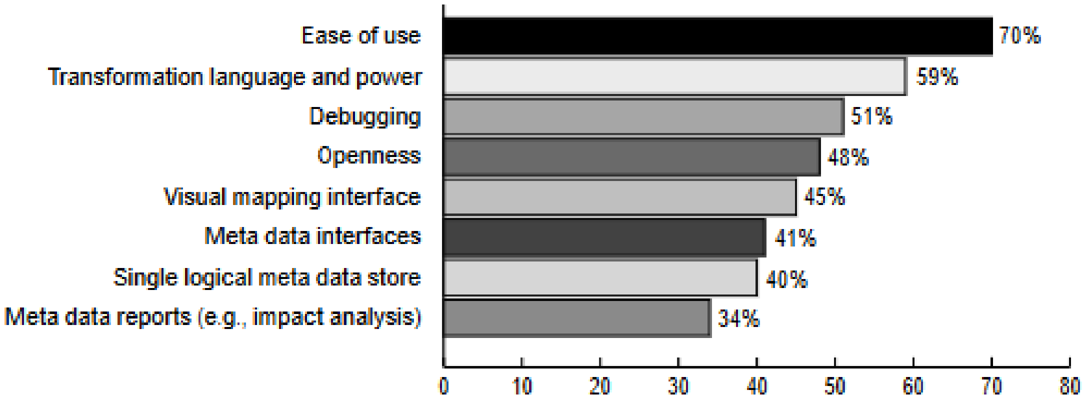

Figure 20: Percentages of survey users who marked the above ETL features as "very important". Percentages are based on responses from 746 participants (Wayne Eckerson & Colin White, 2003).

that was marked "very important" the most by survey participants. Remarkably, the need for *incremental update* or *change data capture* was among the highest ranking *performance features* in terms of importance, as per survey participants votes. This was confirmed even further in the survey section that addresses *extract and load features*, as both features ranked very high in terms of importance out of a set of other features in the same category as figure 19 illustrates. The figure also highlights the importance of an ETL tool's ability to connect and extract data from different data sources and stores, as *breadth of sources* ranked first in importance by a relatively big margin among *extract and load features*. As *breadth of sources* is considered important, also, *breadth of targets supported* is considered important to ETL users and developers want an ETL to support a wide range of target systems. This is in-line with the requirement of an ETL to be a central component in any data-intensive system.

The quality and ease of operation and administration of an ETL is a decisive factor in evaluating its performance and usability. The survey carried out by Wayne Eckerson and Colin White (2003) addressed *operation and administration features* in a separate section. Nearly 80 percent of the participants surveyed in that section marked *error reporting and recovery* as a "very important" administration feature of an ETL. *Debugging* also was marked as "very important" by the majority of respondents. The rankings of these two features show that error handling and recovery process are critical to assessing the quality of ETL administration process. Survey participants want monitoring and managing ETL runtime environment to be efficient and straightforward, with visual consoles and application interfaces. Two thirds of survey respondents marked *Scheduling* as a "very important" feature. Figure 22 shows the rankings of some *operation and administration features* in terms of importance from the perspective of survey

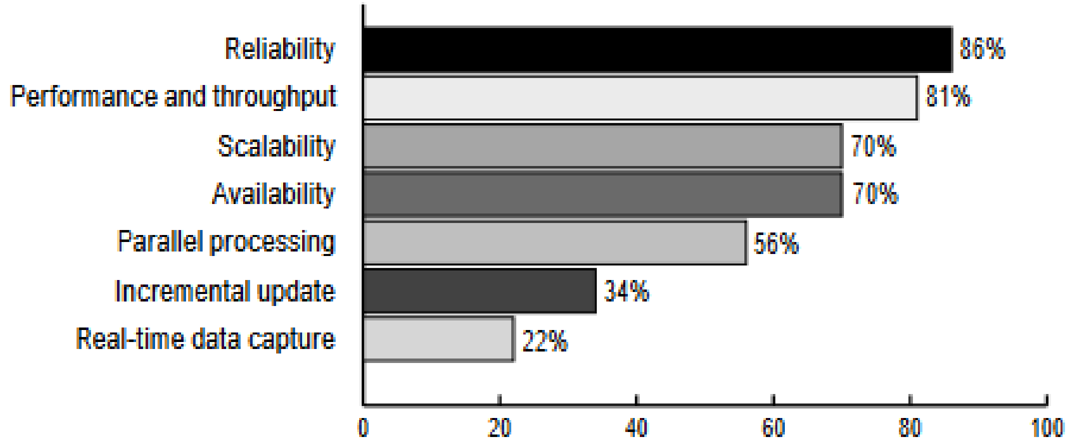

Figure 21: Percentages of survey users who marked the above ETL features as "very important". Percentages are based on responses from 750 participants (Wayne Eckerson & Colin White, 2003).

participants. ETL users and developers want robust, smart, and easy-to-manage ETL schedulers (Wayne Eckerson & Colin White, 2003).

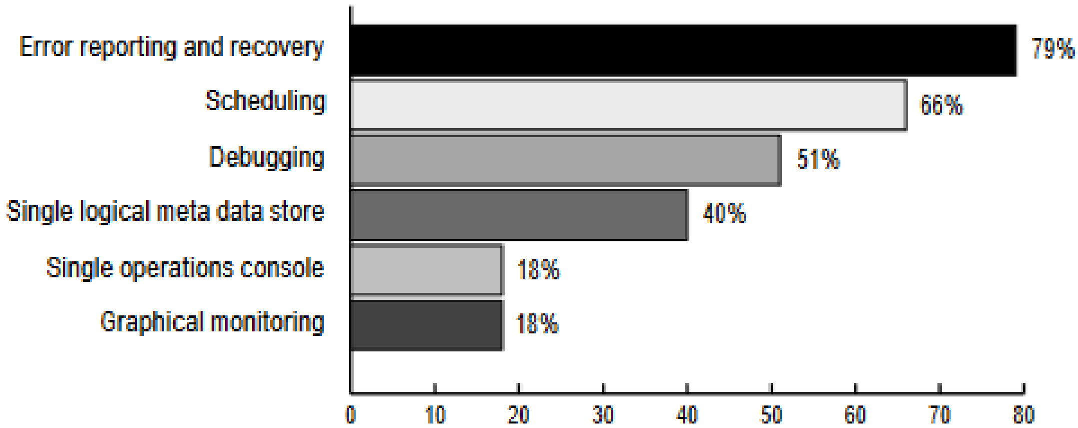

Figure 22: Percentages of survey users who marked the above ETL features as "very important". Percentages are based on responses from 745 participants (Wayne Eckerson & Colin White, 2003).

During this research, it has become clear that a rigid agreed-upon benchmark for evaluating ETL tools is absent so far. It has been noted as well that research on ETL benchmarks and evaluation criteria is relatively scarce.

> The wide spread of industrial and ad-hoc solutions combined with the absence of a mature body of knowledge from the research community is responsible for the absence of a principled foundation of the

fundamental characteristics of ETL workflows and their management (Vassiliadis et al., 2007, p. 2).

Vassiliadis et al. (2007) presented a major step towards a benchmark and test suite for ETL workflows. The proposed benchmark suggested the use of certain quantifiable *measures* to assess ETL tools and methods. Those measures reflect the general desired qualities of an ETL tool. These measures were allocated under four assessment questions. The first assessment question aims at measuring *data freshness and consistency*. As explained in (Vassiliadis et al., 2007), it achieves that through two concrete measures:

- Percentage of data that violate business rules.
- Percentage of data that should be present at their appropriate warehouse targets, but they are not.

The second assessment question addresses *resilience to failures* of the ETL tool. It assesses that quality through abnormal interruption of executions at different stages, and then measuring the percentage of successfully resumed workflow executions. The third assessment question aims at measuring the *speed of the overall process*. As explained in (Vassiliadis et al., 2007), this assessment is carried out using the following measures:

- Throughput of regular workflow execution (this may also be measured as total completion time).
- Throughput of workflow execution including a specific percentage of failures and their resumption.
- Average latency per tuple in regular execution.

The last assessment question addresses *measured overheads* caused by ETL processes execution. Vassiliadis et al. (2007) suggests the following measures for that assessment:

- Min/Max/Avg/ timeline of memory consumed by the ETL process at the source system.
- Time needed to complete the processing of a certain number of OLTP transactions in the presence (as opposed to the absence) of ETL software at the source, in regular source operation.
- Same as the above measure, but in the case of source failure, where ETL tasks are to be performed too, concerning the recovered data.
- Min/Max/Avg/ timeline of memory consumed by the ETL process at the warehouse system.

- (active warehousing) Time needed to complete the processing of a certain number of decision support queries in the presence (as opposed to the absence) of ETL software at the warehouse, in regular operation.
- Same as the above measure, but in the case of any (source or warehouse) failure, where ETL tasks are to be performed too at the warehouse side.

The benchmark proposed in (Vassiliadis et al., 2007) was further updated and expanded in the following years. A direct improvement that included adding more assessment questions and measures was presented in (Simitsis et al., 2009). The improved benchmark modified the previous assessment questions and added new ones. It also updated and increased the measures used to answer each assessment question. The first assessment question remains the same as in the original benchmark. The second assessment question still addressed *resilience to failures*, but included the following measures, as listed in (Simitsis et al., 2009):

- Percentage of successfully resumed workflow executions.
- MTBF, the mean time between failures.
- MTTR, mean time to repair.
- Number of recovery points used.
- Resumption type: synchronous or asynchronous.
- Number of replicated processes (for replication).
- Uptime of ETL process.

The improved benchmark incorporated a new assessment question addressing *maintainability*. As a qualitative aspect, maintainability assessment is not easily achievable through quantitative measures. However, the study in (Simitsis et al., 2009) suggested the following measures to assess an ETL tool's maintainability:

- Length of the longest path in the workflow.
- Complexity of the workflow expressed through the amount of relationships that combine its components.
- *Modularity* (or *cohesion*) refers to the extent to which the workflow components perform exactly one job, thus, a workflow is more modular if it contains less shareable components.
- *Coupling* captures the amount of relationship among different workflow components.

The fourth assessment question in the improved benchmark proposed in (Simitsis et al., 2009) was the same as the third assessment question of the original benchmark in (Vassiliadis et al., 2007). It involved the same three measures as well. The improved benchmark included a fifth assessment question that is meant to address *partitioning*. It evaluates ETL partitioning quality through measurement of the following parameters:

- Partition type.
- Number and length of workflow parts that use partitioning.
- Number of partitions.
- Data volume in each partition.

Another improvement over the original benchmark was the addition of the sixth assessment question that addresses *pipelining*. This is particularly important because it is critical to the potential of parallelization of the ETL workflows. The benchmark in (Simitsis et al., 2009) suggests the following measures to assess the quality of *pipelining* of an ETL tool:

- CPU and memory utilization for pipelining flows or for individual operation run in such flows.
- Min/Max/Avg length of the largest and smaller paths (or subgraphs) containing pipelining operations.
- Min/Max/Avg number of blocking operations.

++++INCOMPLETE+++ Theodorou et al. (2014) presented a model of ETL process quality features and proposed quantitative metrics to assess the degree of absence or existence of each quality. The proposed model is deeply inspired by the work in (Vassiliadis et al., 2007) and (Simitsis et al., 2009). The first desirable quality that an ETL tool should entail, according to the model, is *data quality*, meaning output data quality. It is defined as "the fitness for use of the data produced as the outcome of the ETL process" (Theodorou et al., 2014, p. 8). *data quality*, according to the model, comprises four other important characteristics. The first is *data accuracy*, defined as the percentage of data without data errors. The model proposes two measures

- *data accuracy*, defined as the percentage of data without data errors, and can be measured using the following metric

ETL tools also need to fulfill data compliance requirements. Kimball and Caserta (2011) suggested extensive measures to ensure metadata and data lineage preservation. These measures include archiving snapshots of the data as it passes through the ETL, documentation of the processing and transformation of the data including the algorithms in-place, and keeping proofs of security of the data archives over time. It also stressed the need for the archiving of metadata describing data lineage along with the data itself in all archiving situations. This requirement is in-line with the findings from other research efforts on the topic, as the need for metadata and data lineage management was stressed so frequently. Kimball and Caserta (2011) went over the ETL processes steps and discussed the requirements for each step extensively, and suggested some interesting requirements and desirable qualities. It stressed the need for data profiling capabilities as part of the ETL process. It suggested an increased role of data profiling components in dictating the path of the subsequent workflow steps in the ETL pipeline, as profiling provides insight on how deep the flaws in the data are and how much cleansing is required. Data profiling, as proposed in (Kimball & Caserta, 2011), can even result in the termination of the respective ETL workflow, if the data is deemed unfit for the business objective. The work in (Kimball & Caserta, 2011) stressed the need for improving end-user delivery interfaces as core factor in data usability. It states that data should be handed to the end-user application in a way that does not add complexity to the application. In fact, it required data to be delivered through an interface that improves the speed and simplicity of the end-user application. It considered it "irresponsible" to introduce unnecessary complexity or latency to end users. This attention to data understanding and end-user convenience is consistent with the popular *dimensional modelling* technique that is widely adopted in data warehouses and systems design (Kimball, 1997).

The work in (Kimball& Caserta,2011) urged for increased focus on data quality through ETL processes. It describes *competing factors* that dictate the priorities of the data quality assurance components of an ETL. Figure 23 shows the four *competing factors* as described in (Kimball & Caserta,2011). A data quality or cleansing subsystem in an ETL is required to be thorough in order to deliver reliable data, but this comes at the expense of speed. The data cleansing component of an ETL also needs to ensure high performance and speed to be able to process the ever-increasing amounts of data that pass through. This highlights the need for thorough but optimized data cleansing subsystems of ETL solutions. Data quality assurance also requires corrective measures to be applied to the incoming data. Data quality issues need to be corrected and addressed, but this comes at the expense of transparency. Extensive masking and remedying of data quality issues in an organization’s data can be harmful as it allow data quality issues at the source to foster for years without notice or reporting. This stresses the need for corrective but transparent data quality and cleansing operations in ETL workflows(Kimball & Caserta, 2011).

The importance of *change data capture* to ETL tools efficiency has been confirmed throughout most of the literature addressing ETL qualities and design process. Randal et al. (2011) discussed ETL systems design, and stressed certain

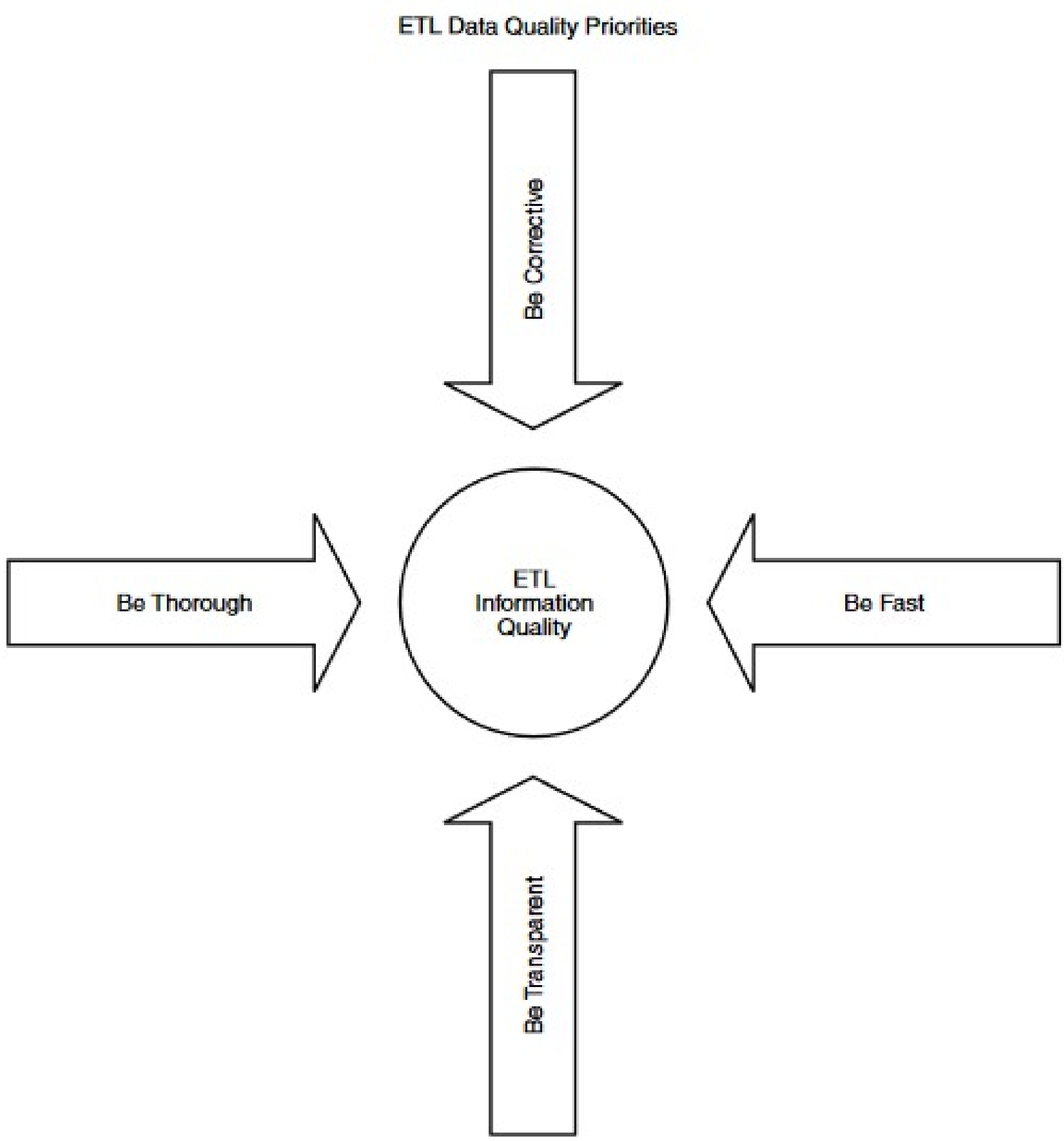

Figure 23: Data quality priorities (Kimball & Caserta, 2011).

qualities as necessary for a modern ETL system design. Central to those qualities was the choice of *change data capture* techniques that an ETL utilizes in order to detect changed data. It reasoned for the importance of *change data capture* as it achieves minimum extraction, processing, and loading volume, increases system performance, and reduces the risk of duplicate insertions or updates. Another desirable quality that was highlighted in (Randal et al., 2011) is *recovery and re-start ability*. ETL systems are required to be able to recover from errors, restart, or retry without causing inconsistency in the output data or requiring complex data cleanup procedures. "ETL processes should be *re-entrant*. This means they can be executed a second time, even after a failure, without posting duplicate transactions or skipping unprocessed rows" (Randal et al., 2011, p. 534). *Recoverability* can also be enhanced by introducing *redundancy* in the ETL workflow processes which can improve *resilience to failures* (Simitsis et al., 2009). *Redundancy* "can be achieved with three techniques: replication, diversity or fail-over" (Simitsis et al., 2009, p. 9).

There are shortcomings or missing features that are more common in *open source* ETL tools than commercial or *closed source* ones. This makes it more important to stress these qualities and features in the design and development of any *open source* ETL tool. Kabiri and Dalila (2013) carried out a survey of ETL tools varying between being commercial products, open source tools, and research prototypes. The survey highlighted the need for *open source* ETL tools to incorporate the ability to load multidimensional cubes, or ROLAP and MOLAP capabilities. It also stressed the need for having *incremental update* or *change data capture* capabilities in order for *open source* ETL tools to be usable in wider scope of applications. The survey also highlighted the need for *open source* ETL tools to provide more low-code or graphical interfaces to enhance *ease of use*. Another important aspect of *open source* ETL tools that is critical for its successful adoption is the size of users' community. The survey also stressed the need for more comprehensive documentation and active support for *open source* ETL tools to be more widely usable. Another critical factor of success for *open source* ETL tools is their ability to integrate and connect to other Business Intelligence (BI) suites and tools (Kabiri & Dalila, 2013).

C A walkthrough of ODS GUI and API Functionality

The ODS can be used either through its REST API or through its GUI. Each interface allows access to a set of features. We begin by examining the GUI which is served through the *Web-Client* and *Reverse-Proxy* microservices. The GUI *home page* indicates that it is dedicated for displaying a dashboard. However, the dashboard view seems not to be implemented. The *home page* shows only a welcome banner, and a side panel that can be used to navigate to other pages as shown in figure 24. We can conclude that the ODS has a *user interface*, and is partially manageable through that *user interface*. It is important to note here that the web GUI of the ODS does not restrict users actions into a wizard-like path. For example, a user can start a workflow by navigating directly to the *pipelines* creation and management page. Creating a pipeline that has no defined *data source* is possible through the ODS GUI. However, the user will have to manually provide that data, in JSON format, through a text input element.

The first logical step in a workflow through ODS GUI starts from the *data sources* management page. The page provides an overview of the *data sources* that have been configured in the ODS so far, either through the API or the GUI. At the top level, the page displays the most important information and metadata corresponding to each data source. Alongside each data source, there exists a button that shows and hides a collapsible panel that contain a more complete overview of the metadata related to the data source. A data source can be defined, deleted, edited, or triggered through this page. The metadata of a data source can be defined, or edited through this page as well. The data source configuration interface has a section for configuring the *adapter* service for the data source, which allows for defining the protocol, format, location, and encoding of the data source, as shown in figure 26. There is also a section in the data source configuration interface that allows for adjusting the intervals at which the data should be periodically fetched, as shown in figure 27. There exists a button, alongside each data source, that leads to a pipeline creation interface so that the user can create a pipeline specifically for that data source without having to provide data source identification details in the pipeline creation panel. The page also provides a search bar that can be used to look up a data source using its name. Figure 25 shows the *data sources* page with two example data sources defined for demonstration purposes. The metadata panel of the first data source is expanded to demonstrate that state.

The second step after creating a *data source* is to create or define a transformation pipeline. The GUI provides a page for *pipelines* management. Figure 28 shows the page for *pipelines* management with three example pipelines created for demonstration. The first of these pipelines is created without being connected to a configured data source. As mentioned earlier, creation of a pipeline does not necessarily require a data source. However, this requires manual entry of input data. The normal process starts by opening the pipeline creation interface by clicking on the button labeled "CREATE NEW PIPELINE". This leads to a page that queries the user for the relevant entries to define the pipeline. The most important entry is the data source id, as it allows for showing an excerpt of the data fetched from the data source in order to live-test the defined transformations, as shown in figure 29. The pipeline creation interface also allows for editing metadata of the pipeline, as it has editable entries for name, data source id, description, author, and license of the created pipeline. Pipelines can be created, edited, viewed, looked up, and deleted through this page of the GUI. There is a droplet symbol to the right of the page alongside each pipeline that shows the "status" of the pipeline, and turns green when the pipeline has output data. For each pipeline, there exists a button with an alarm symbol that leads to *notification* creation interface. Alongside each pipeline entry in the page, there exists a button with a storage disk symbol that leads to a data view page that displays all the data that has been processed by the pipeline so far. An example of the data view page is shown in figure 30.

As mentioned earlier, the *notification* service is also manageable through the GUI. It can be reached through the *pipelines* management page. The *notifications* management interface for each pipeline separately can be reached by clicking the alarm-shaped button alongside its entry in the *pipelines* management page. Figure 31 shows the notifications management page for a certain pipeline with three notifications set-up for demonstration purposes. The page allows for creating new notifications for the pipeline and managing previously created ones. The notification creation interface requires input of certain parameters according to which notification method is chosen by the user. The available modes are:

- Firebase Cloud Messaging (FCM)
- Webhook
- Slack

While using the ODS through the *user interface* seem to cover a lot of functionality to manage an ETL workflow, the ODS is manageable through an Application Programming Interface (API) as well. As indicated by the current architecture of ODS, shown in figure 3.1, ODS v2 was transformed from a *monolith* software into a *microservices* architectural style, as explained in detail in (Schwarz, 2019). The functionality in ODS v2 is carried out by the following six *microservices*:

- *Datasource* service

- *Pipeline* service
- *Scheduler* service
- *Query* service
- *Web client* service
- *Notification* service
- *Reverse proxy* service

Out of the above services, only the *datasource*, *pipeline*, *query*, and *notification* services are partially exposed to the user through the API. The *Web client*, and *scheduler* services are not available to the user through the API, as they are meant to support other services internally.

The *data source* API exposes several endpoints to provide access to functionality for *adapter* and *data sources* configuration. There are some terms attributed to the *data source* service that need to be clarified in order to understand its functionality. *Data source* in the context of the ODS is a data source that has been configured and defined into the *data source* service by providing protocol, location, encoding, format, and other information. *Protocol configuration* is a set of parameters that contain the minimum required information about a data source such as location, protocol, and encoding. *Adapter configuration* is a set of parameters that include *protocol configuration* information, as well as information about format type and parameters. *Preview* is a one-time fetching of data without necessarily defining a *data source*, and is carried out according to a *protocol configuration* alone, which is then called *raw preview*, or according to the more comprehensive *adapter configuration*. A *preview* results in no downstream processing of the data. A *data import*, to the contrary, is a one-time data import that is attributed to a data source and gets passed to the *query* service. It is worth noting that deletion of *data sources* through the *data source* API does not result in deletion of any *pipelines* defined on top of them. The user has to delete *pipelines* separately. The *data source* service API allows the user to execute the following functionality:

- get service version
- get supported data formats
- get supported data transfer protocols
- execute a *preview* and receive the fetched data in the response
- get all *data sources* configurations
- get configuration of a single *data source*

- create a *data source*
- edit and update a *data source*
- delete a single *data source*
- delete all *data sources*
- define dynamic parameters in a *data source* configuration that can be used to import data slices
- trigger *data import* with or without parameters
- fetch all *data imports* of a *data source*
- fetch a single *data import* of a *data source* using the *data import* id
- fetch the latest *data import* of a *data source*
- fetch the content data of a *data import* using the *data import* id
- fetch the content data of the latest *data import* of a *data source*

The *pipeline* service exposes several endpoints to allow the user to create and configure data transformation pipelines through the API. Some terms and constructs related to the *pipeline* API need to be clairfied first in order to understand the functionality of the service. A *pipeline execution request* is a construct that contains a data set and a data transformation script. It is used across the API to execute one-off data transformation jobs. A *pipeline configuration trigger request* is a construct that contains a data set and a *data source* id. It is used for triggering data transformation pipelines associated with a certain *data source* and passing the new data batch along. A *pipeline configuration* is a set of parameters that outline the main attributes of a *pipeline* such as *pipeline* id, id of the associated *data source*, transformation script, *pipeline* author, display name, license, description, and creation timestamp. The *pipeline* service API allows the user to execute the following functionality:

- get health status of the service
- get service version
- execute one-off data transformation jobs and get transformed data, error report, and process statistics in the response
- trigger pipelines with new data batches using *data source* id
- get all *pipeline configurations*
- get a single *pipeline configuration*
- create a *pipeline configuration*

- update a *pipeline configuration*
- delete all *pipeline configurations*
- delete a single *pipeline configuration*

As data retrieval through the ODS GUI is not practically usable, the *query* (or *storage*) service carries out a critical role in allowing the user to retrieve and use data that has been processed by the ETL workflow. The *query* service exposes several endpoints that allow the user to execute the following functionality:

- create storage *structure* for the storage of data from a certain *pipeline*
- delete storage *structure* of a *pipeline*
- post *pipeline* data for storage
- get stored data of a *pipeline*

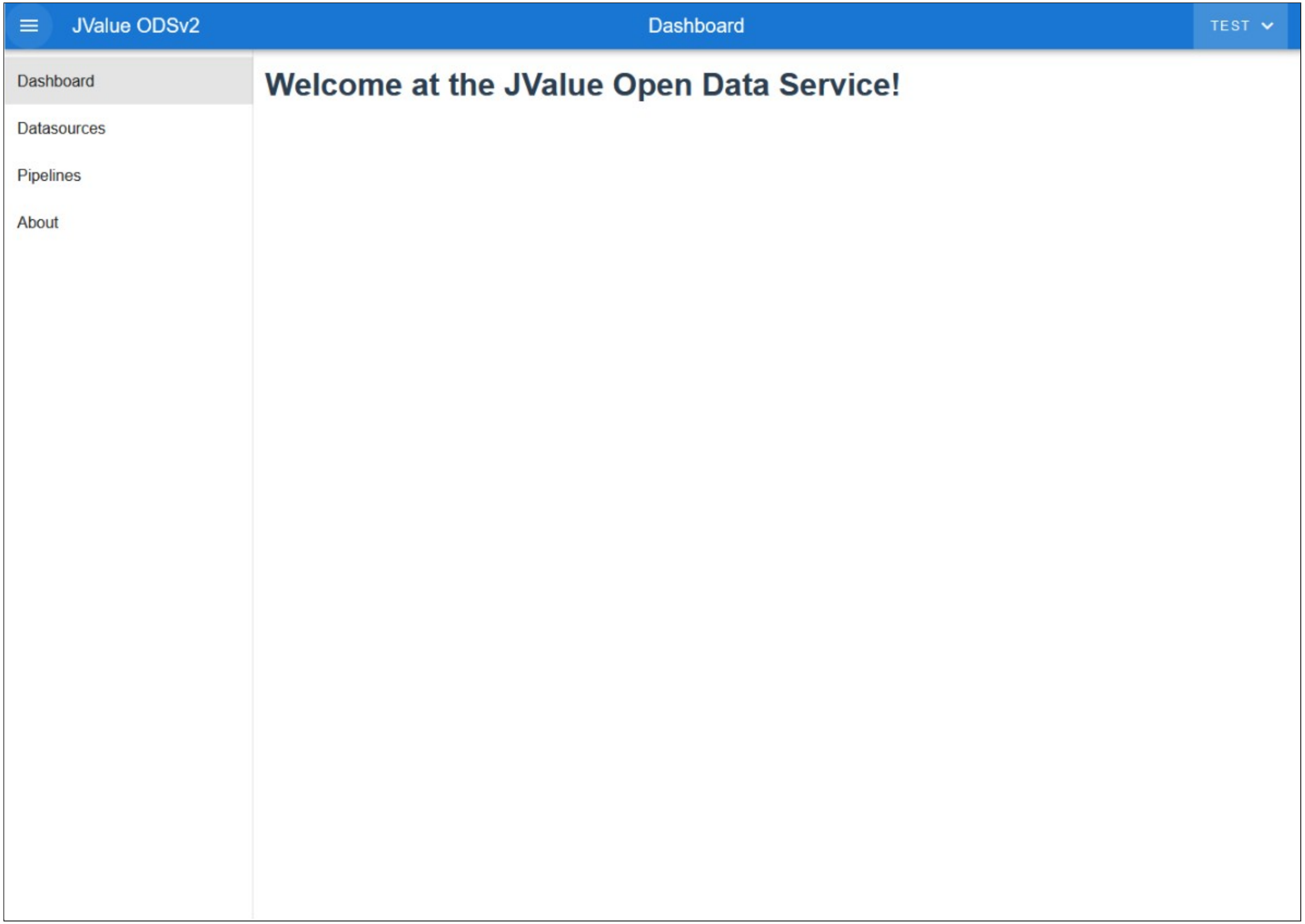

Figure 24: *Home page* of the web GUI of the ODS.

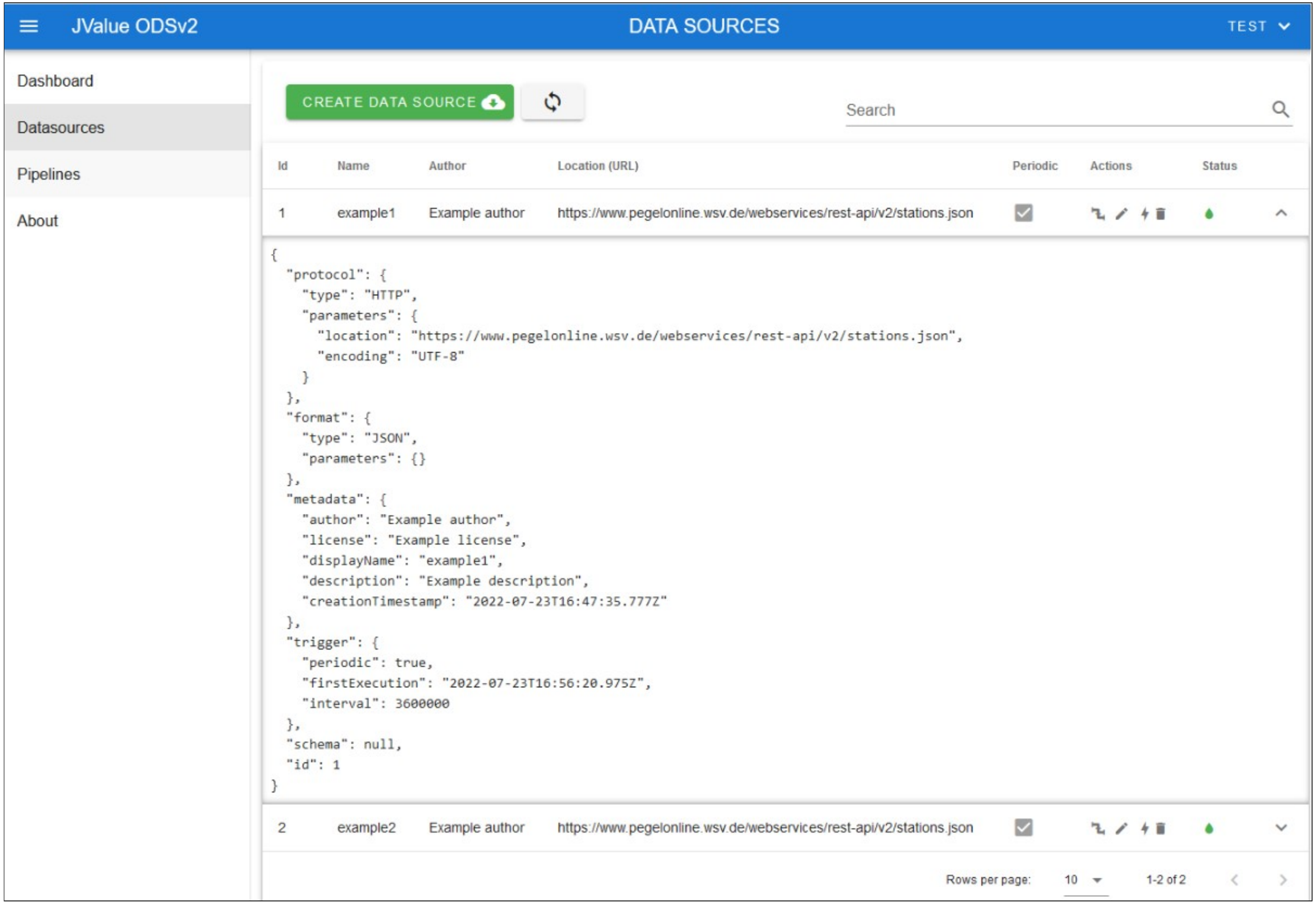

Figure 25: The page for *data sources* management in the web GUI of the ODS.

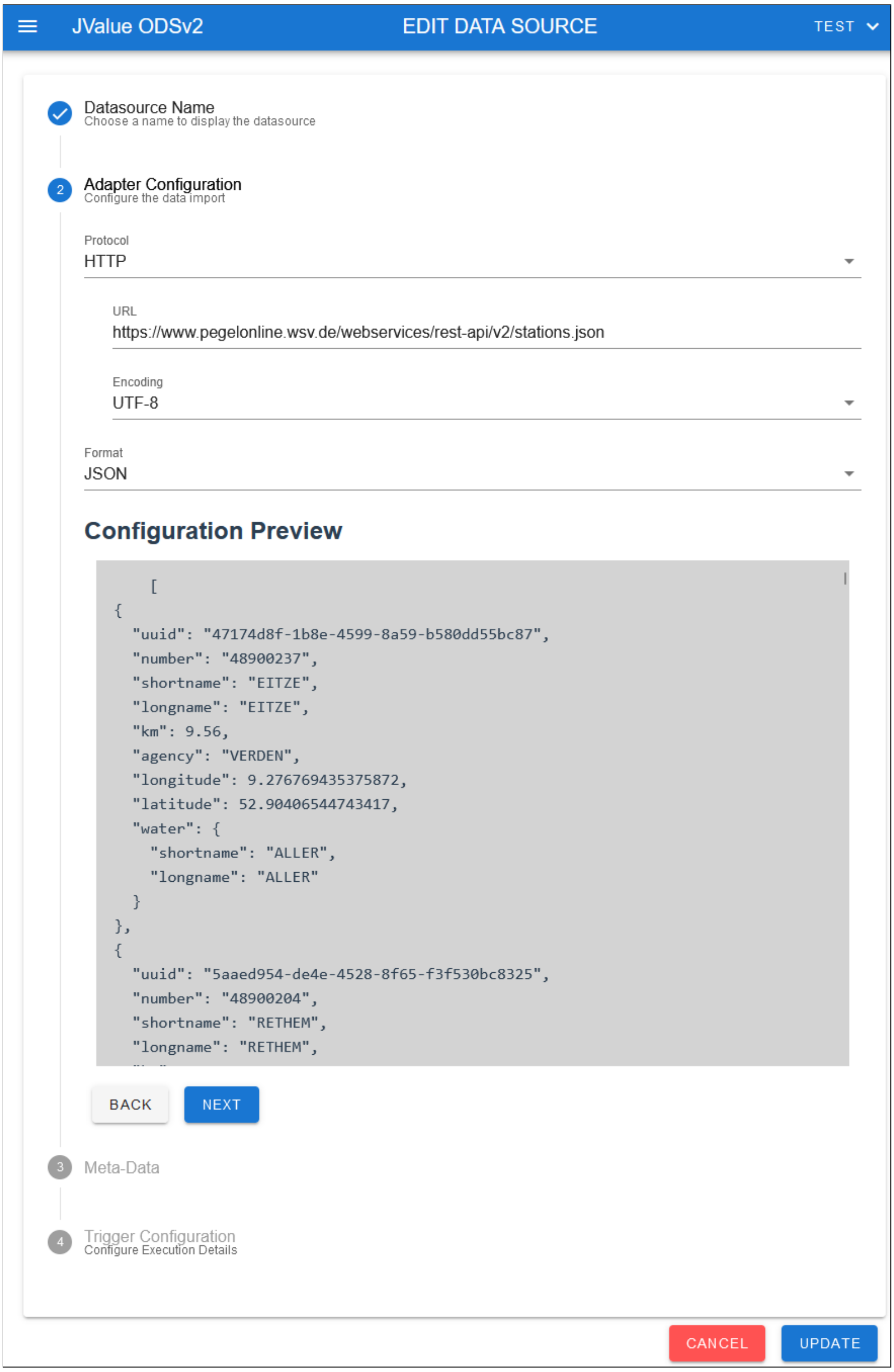

Figure 26: Data source configuration interface in the web GUI of the ODS contains a section for configuring the adapter service with the data source characteristics.

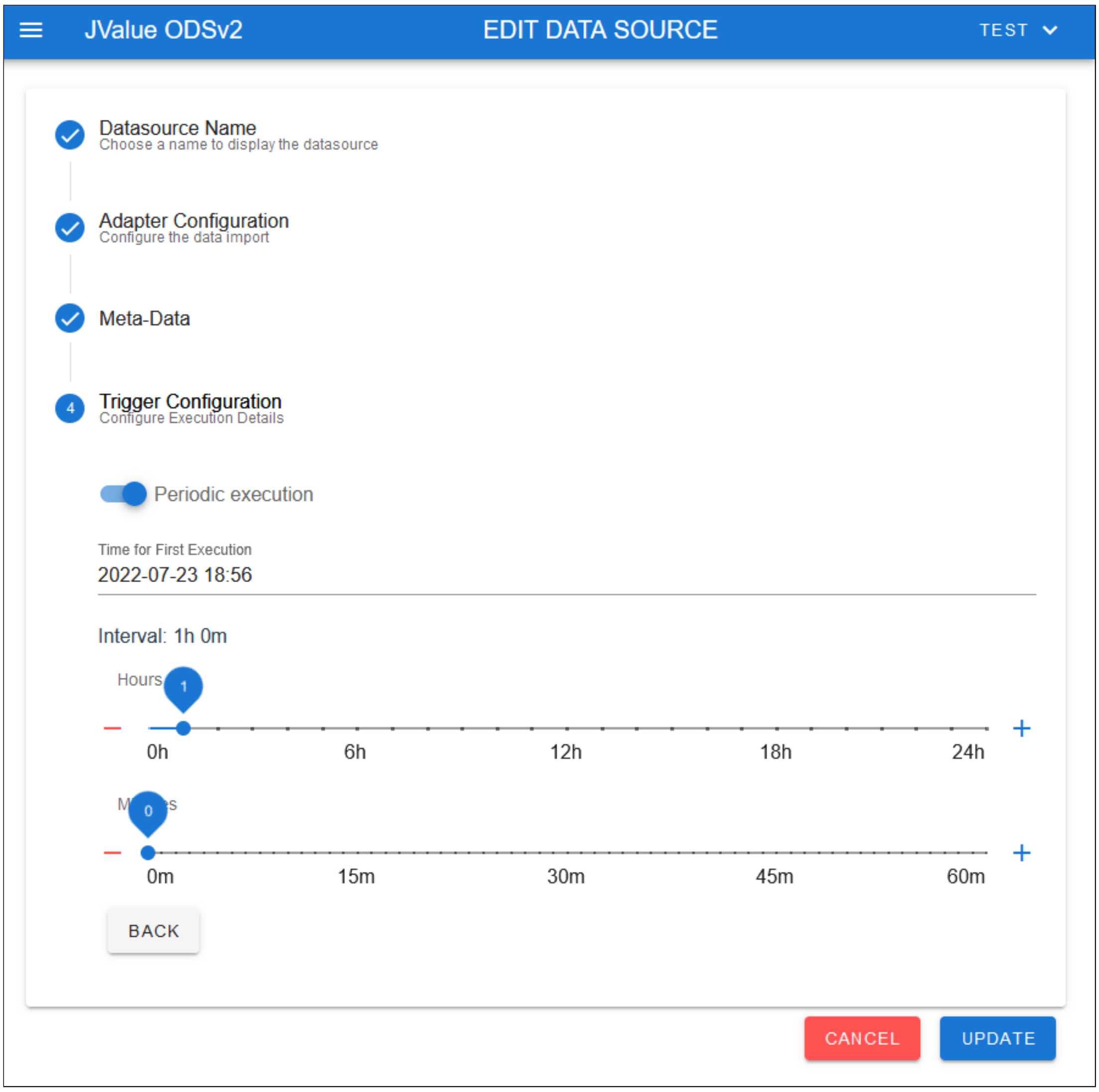

Figure 27: Data source configuration interface in the web GUI of the ODS contains a section for adjusting periodic data fetching intervals.

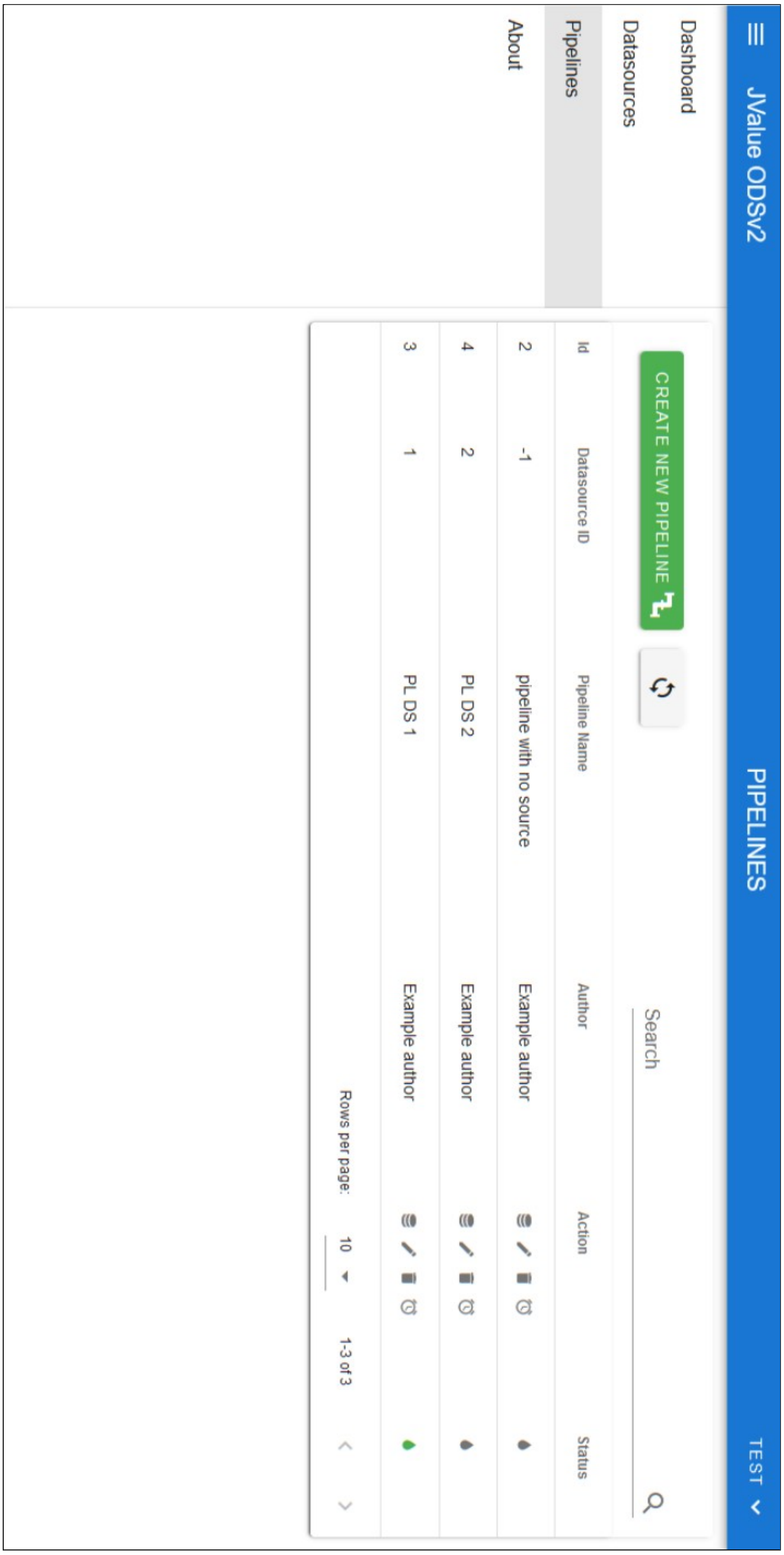

Figure 28: The page for *pipelines* management in the web GUI of the ODS.

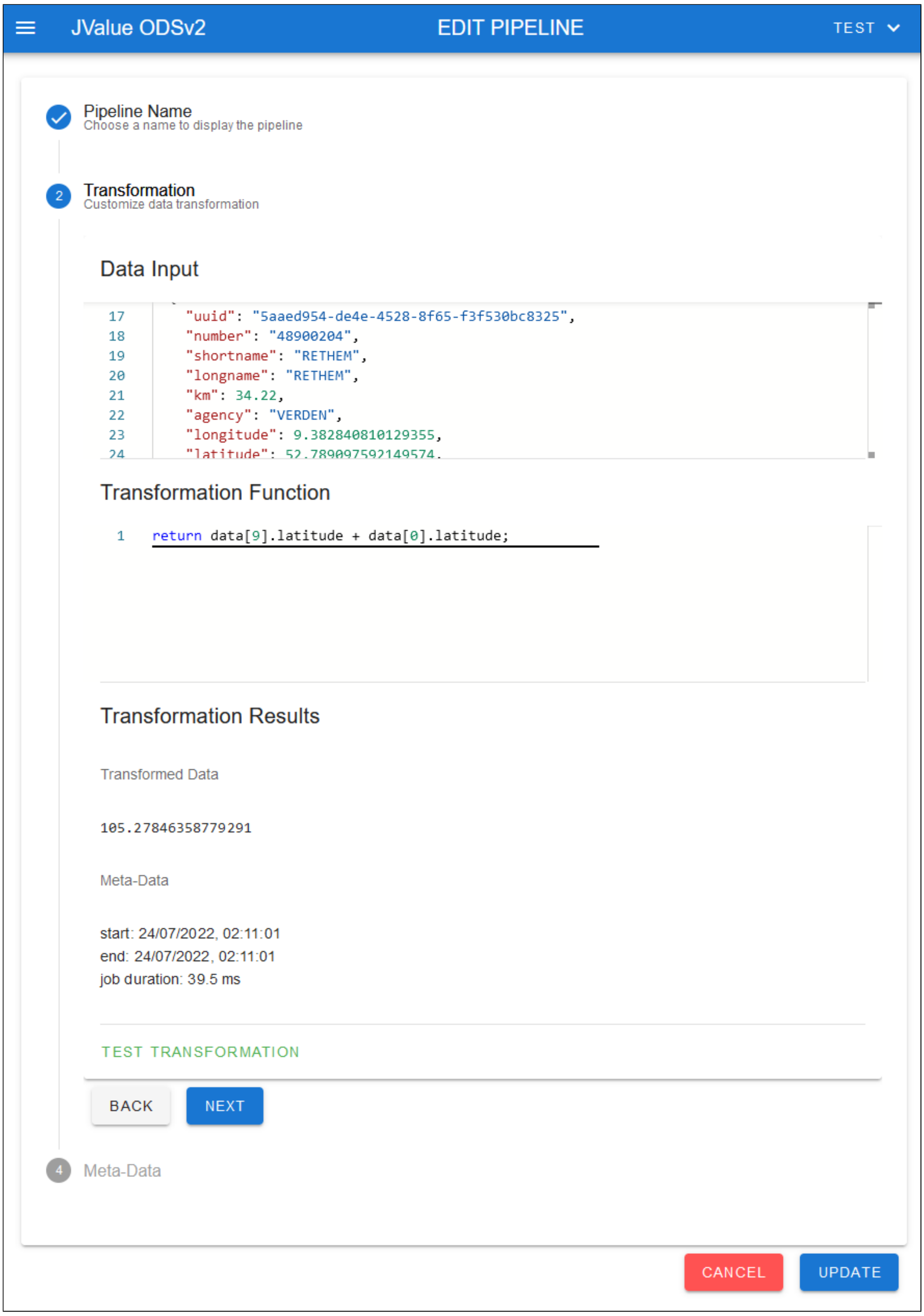

Figure 29: Pipeline creation interface in the web GUI of the ODS contains a section for defining data transformations.

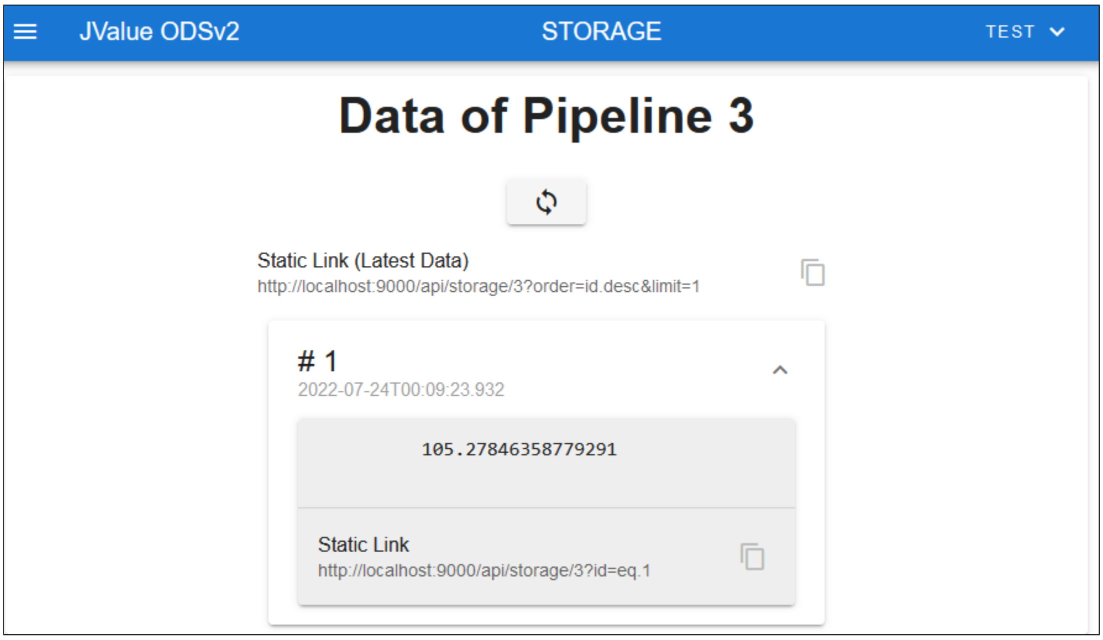

Figure 30: Processed data of each pipeline can be accessed through the *pipelines* management page in the web GUI of the ODS.

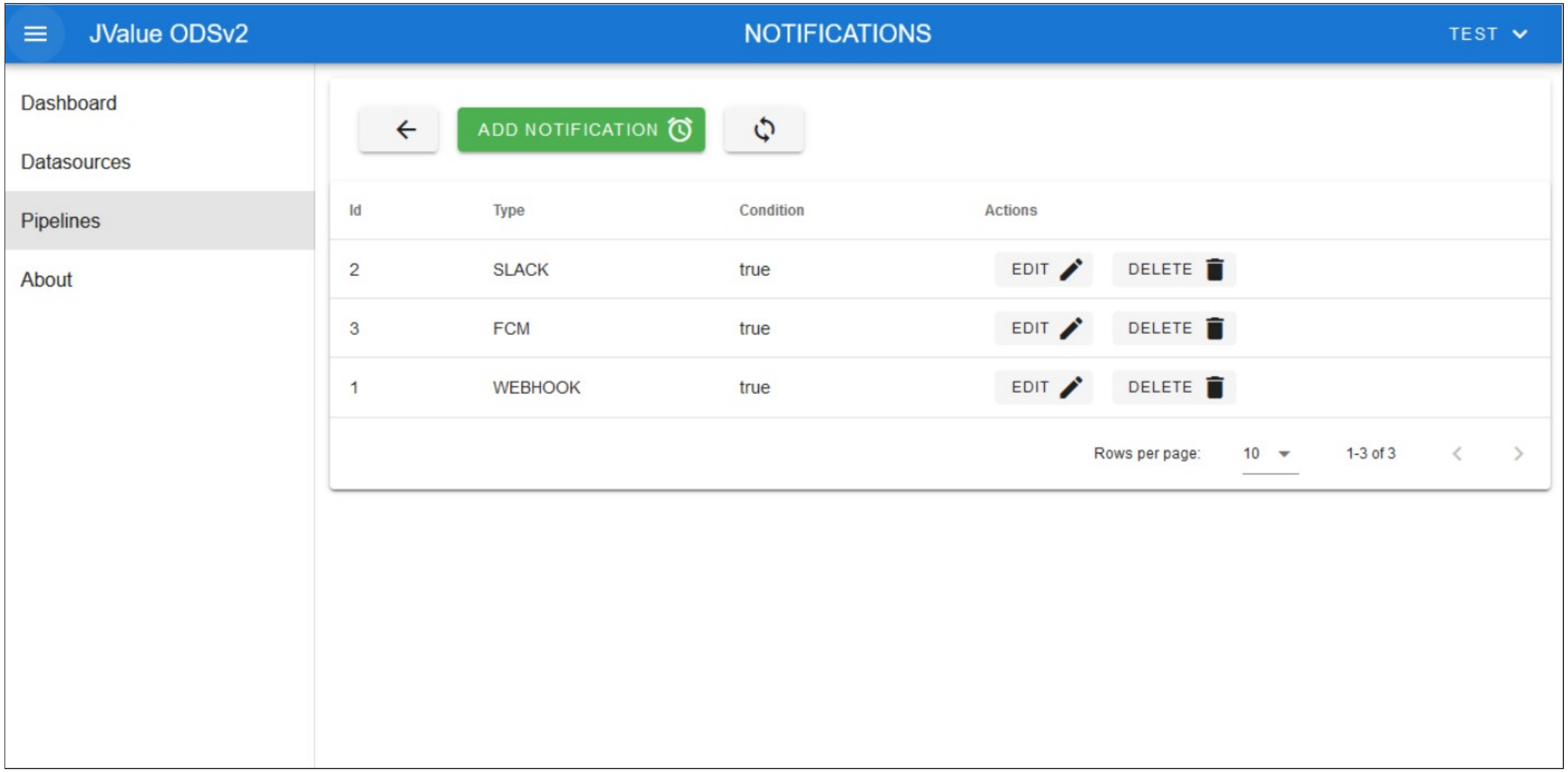

Figure 31: Notifications for each pipeline can be managed through a separate page in the web GUI of the ODS.

References

Albrecht, A. & Naumann, F. (2009). Managing ETL processes.

Anaconda Inc. (2021). *Anaconda | State of Data Science 2021*. Anaconda. Retrieved June 23, 2022, from https://www.anaconda.com/state-of-data-science-2021

Ankorion, I. (2005). Change Data Capture Efficient ETL for Real-Time BI. *DM Review*, *15* (1), 36. Retrieved June 28, 2022, from https://www.proquest.com/docview/214690875/abstract/5201ABD8EF3A4723PQ/1

Berners-Lee, T. (2006). *Linked Data - Design Issues*. Retrieved June 14, 2022, from https://www.w3.org/DesignIssues/LinkedData.html

Cao, L. (2017). Data Science: A Comprehensive Overview. *ACM Computing Surveys*, *50*, 1–42. https://doi.org/10.1145/3076253

Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C. & Wirth, R. (2000). CRISP-DM 1.0: Step-by-step data mining guide. *undefined*. Retrieved June 25, 2022, from https://www.semanticscholar.org/paper/CRISP-DM-1.0%3A-Step-by-step-data-mining-guide-Chapman-Clinton/54bad20bbc7938991bf34f86dde0babfbd2d5a72

Concilio, G. & Molinari, F. (2021). The Unexploitable Smartness of Open Data. *Sustainability*, *13*, 8239. https://doi.org/10.3390/su13158239

CrowdFlower. (2016). *CrowdFlower_DataScienceReport_2016.pdf*. Retrieved June 16, 2022, from https://visit.figure- eight.com/rs/416- ZBE- 142/images/CrowdFlower_DataScienceReport_2016.pdf

Eden, A. & Kazman, R. (2003). Architecture, Design, Implementation., 149–159. https://doi.org/10.1109/ICSE.2003.1201196

Fayyad, U., Piatetsky-Shapiro, G. & Smyth, P. (1996, February 1). *From data mining to knowledge discovery: An overview* (Vol. 17).

Gertosio, C. & Dussauchoy, A. (2004). Knowledge discovery from industrial databases. *Journal of Intelligent Manufacturing*, *15*, 29–37. https://doi.org/10.1023/B:JIMS.0000010073.54241.e7

Global Report | Open Data Barometer. (2017). Retrieved June 20, 2022, from https://opendatabarometer.org/4thedition/report/

Gray, J. (2014, September 3). Towards a Genealogy of Open Data. https://doi.org/10.2139/ssrn.2605828

Hamilton, H. (2000). *KDD Process/Overview*. Retrieved June 24, 2022, from http://www2.cs.uregina.ca/~dbd/cs831/notes/kdd/1_kdd.html

Hickmann Klein, R., Barbiero Klein, D. & M Luciano, E. (2017, October 1). *Open Government Data Concept Over Time: Approaches and Dimensions*.

Jvalue Project. (2022, April 2). *Open Data Service (ODS)*. Retrieved July 23, 2022, from https://github.com/jvalue/ods

Kabiri, A. & Dalila, C. (2013). Survey on ETL processes. *Journal of Theoretical and Applied Information Technology*, *Vol. 53*.

Kaggle Inc. (2022, July 26). *Kaggle Competitions*. Retrieved July 26, 2022, from https://www.kaggle.com/competitions?sortOption=numTeams

Kimball, R. (1997, August 2). *A Dimensional Modeling Manifesto*. Kimball Group. Retrieved June 10, 2022, from https://www.kimballgroup.com/1997/08/a-dimensional-modeling-manifesto/

Kimball, R. & Caserta, J. (2011). *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data.* John Wiley & Sons, Inc. Retrieved June 28, 2022, from http://www.SLQ.eblib.com.au/patron/FullRecord.aspx?p=219051
OCLC: 780986568

Kurgan, L. & Musilek, P. (2006). A survey of Knowledge Discovery and Data Mining process models. *Knowledge Eng. Review*, *21*, 1–24. https://doi.org/10.1017/S0269888906000737

Kvamsdal, P. (2017, December 1). *Open Government Data - A Literature Review and a Research Agenda*.

Mariscal, G., Marbán, O. & Fernández, C. (2010). A survey of data mining and knowledge discovery process models and methodologies. *Knowledge Eng. Review*, *25*, 137–166. https://doi.org/10.1017/S0269888910000032

Martinez, I., Viles, E. & Olaizola, I. (2021). Data Science Methodologies: Current Challenges and Future Approaches. *Big Data Research*, *24*, 100183. https://doi.org/10.1016/j.bdr.2020.100183

Martinez-Plumed, F., Contreras-Ochando, L., Ferri, C., Hernandez-Orallo, J., Kull, M., Lachiche, N., Ramirez-Quintana, M. J. & Flach, P. (2021). CRISP-DM Twenty Years Later: From Data Mining Processes to Data Science Trajectories. *IEEE Transactions on Knowledge and Data Engineering*, *33* (8), 3048–3061. https://doi.org/10.1109/TKDE.2019.2962680

Microsoft. (2022). *What is the Team Data Science Process? - Azure Architecture Center*. Retrieved June 22, 2022, from https://docs.microsoft.com/en-us/azure/architecture/data-science-process/overview

Mons, B., Schultes, E., Liu, F. & Jacobsen, A. (2020). The FAIR Principles: First Generation Implementation Choices and Challenges. *Data Intelligence*, *2* (1-2), 1–9. https://doi.org/10.1162/dint_e_00023

Moyle, S. & Jorge, A. (2001). RAMSYS-A methodology for supporting rapid remote collaborative data mining projects.

Piovesan, F. (2015, July 7). *Beyond Standards and Regulations: Obstacles to Local Open Government Data Initiatives in Italy and France*. https://doi.org/10.13140/RG.2.1.3572.5528

Randal, P. S., Tripp, K., Nielsen, P. & Delaney, K. (2011, October 12). *SQL Server MVP Deep Dives, Volume 2*. Simon and Schuster.

Riehle, D. (2011). Controlling and Steering Open Source Projects. *Computer*, *44* (7), 93–96. https://doi.org/10.1109/MC.2011.206

Riehle, D. (2015). The Five Stages of Open Source Volunteering. In W. Li, M. N. Huhns, W.-T. Tsai & W. Wu (Eds.), *Crowdsourcing: Cloud-Based Software Development* (pp. 25–38). Springer. https://doi.org/10.1007/978-3-662-47011-4_2

Riehle, D. (2019a, September 4). *Enabling Open Innovation with Open Data using the JValue Open Data Service*. Software Research and the Industry. Retrieved June 21, 2022, from https://dirkriehle.com/2019/09/04/enabling-open-innovation-with-open-data-using-the-jvalue-open-data-service/

Riehle, D. (2019b). The Innovations of Open Source. *Computer*, *52*, 59–63. https://doi.org/10.1109/MC.2019.2898163

Robinson, P. & Scassa, T. (2022). *The Future of Open Data*. Retrieved June 11, 2022, from http://ruor.uottawa.ca/handle/10393/43648
Accepted: 2022-05-25T14:08:26Z

Saltz, J. (2020, November 30). *CRISP-DM is Still the Most Popular Framework for Executing Data Science Projects*. Data Science Process Alliance. Retrieved June 24, 2022, from https://www.datascience-pm.com/crisp-dm-still-most-popular/

SAS Institute. (2012, March 8). *SAS Institute: SEMMA*. Retrieved June 25, 2022, from https://web.archive.org/web/20120308165638/http://www.sas.com/offices/europe/uk/technologies/analytics/datamining/miner/semma.html/

SAS Institute. (2017). *SAS Help Center: Introduction to SEMMA*. Retrieved June 25, 2022, from https://documentation.sas.com/doc/en/emref/14.3/n061bzurmej4j3n1jnj8bbjjm1a2.htm

Schwarz, G.-D. (2019). Migrating the JValue ODS, 78.

Simitsis, A., Vassiliadis, P., Dayal, U., Karagiannis, A. & Tziovara, V. (2009, August 24). *Benchmarking ETL workflows* (Vol. 5895). https://doi.org/10.1007/978-3-642-10424-4_15

Theodorou, V., Abelló, A. & Lehner, W. (2014). Quality Measures for ETL Processes. https://doi.org/10.1007/978-3-319-10160-6_2

Theodorou, V., Abelló, A., Lehner, W. & Thiele, M. (2015). Quality measures for ETL processes: From goals to implementation. *Concurrency and Computation Practice and Experience*, *28*, http://onlinelibrary.wiley.com/doi/10.1002/cp https://doi.org/10.1002/cpe.3729

Two Crows Corporation. (1999). *Introduction to Data Mining and Knowledge Discovery*. Two Crows Corporation.

Vassiliadis, P., Karagiannis, A., Tziovara, V. & Simitsis, A. (2007, January 1). *Towards a Benchmark for ETL Workflows.*
Vassiliadis, P. & Simitsis, A. (2009). Extraction, transformation, and loading. *Encyclopedia of Database Systems*.
Vassiliadis, P., Simitsis, A. & Skiadopoulos, S. (2002). Conceptual Modeling for ETL Processes. https://doi.org/10.1145/583890.583893
Vetro, A., Canova, L., Torchiano, M., Minotas, C., Iemma, R. & Morando, F. (2016). Open data quality measurement framework: Definition and application to Open Government Data. *Government Information Quarterly*, *33*. https://doi.org/10.1016/j.giq.2016.02.001
Vetro, A., Torchiano, M., Orozco, C., Procaccianti, G., Iemma, R. & Morando, F. (2014). An Exploratory Empirical Assessment of Italian Open Government Data Quality With an eye to enabling linked open data. *http://softeng. polito. it/reports/Softeng_TechReport_2014. pdf*.
Wayne Eckerson & Colin White. (2003). *Evaluating ETL and Data Integration Platforms*. Data Warehousing Institute (TDWI).
Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L. B., Bourne, P. E., Bouwman, J., Brookes, A. J., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., Evelo, C. T., Finkers, R., . . . Mons, B. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, *3* (1), 160018. https://doi.org/10.1038/sdata.2016.18