Using Apache Spark for Ensuring Data Quality in Modern Data Lake Pipeline Architectures

Martina Šestak[1,*,†], Timi Vovk[1,†]

[1]University of Maribor, Faculty of Electrical Engineering and Computer Science, Koroška cesta 46, 2000 Maribor, Slovenia

Abstract

The large volumes of data generated in modern IT systems raise the need for data quality control mechanisms to ensure that business decisions are grounded in high-quality and accurate data. Big data systems face additional data quality-related challenges that must be addressed throughout the entire data processing pipeline. In this paper, we focus on ensuring data quality in data lakes, as their flexibility in handling heterogeneous data is often left unaccompanied with suitable data quality mechanisms, leading to data swamps and poor results overall. We explore the Apache Spark framework and its features that can be used to implement data lake pipelines while ensuring data quality. A data lake implementation for an agri-food use case based on Spark confirms the suitability of the framework for implementing ELT (Extract, Load, Transform) and other processing jobs in a data lake, data pipeline orchestration and running tests to maintain data quality in data lakes.

Keywords

Big data quality, Data lake, Apache Spark, Unit testing

1. Introduction

Modern big data environments are often designed to handle data obtained from heterogeneous sources, each with its own degree of data structure. This data serves as a valuable asset exchanged between various information systems within this environment. However, if left unsupervised, its quality can quickly deteriorate over time. Consequently, the main challenges in maintaining big data quality are the large volumes of data generated by modern IT systems, heterogeneity, the constant changes in data, and data security [1].

Data quality is commonly defined as "the degree to which data satisfy the requirements defined by the product-owner organization" [2]. In the context of high data volumes, ensuring data quality becomes increasingly critical to facilitate accurate data-driven decision-making. When the quality of the data used for decision-making is poor, it often leads to incorrect findings, erroneous decisions, sub-optimal process executions, revenue losses, and other associated costs [3].

SQAMIA 2023: Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications, September 10–13, 2023, Bratislava, Slovakia

*Corresponding author.

†These authors contributed equally.

martina.sestak@um.si (M. Šestak); timi.vovk1@um.si (T. Vovk)
0000-0001-7054-4925 (M. Šestak)

Poor data quality can arise from several reasons, such as errors during data entry, using inappropriate data collection methods, failure to update dynamic data, misapplication of business rules, presence of duplicate records, and missing or incorrect data values [4]. These mistakes introduce anomalies in data quality, which decreases its suitability for data-driven decision-making. Low levels of data quality significantly impact the overall effectiveness of data applications. As data volumes and varieties increase, controlling the quality of each data entry becomes more challenging. The variety and volume of big data introduce additional complexities to traditional data quality dimensions, such as accuracy and completeness [5]. The real-time nature of data, such as streaming data or IoT (Internet of Things) data, further complicates data quality assessment [6].

Recently, data lakes have emerged as a viable solution for establishing a centralized storage system that can accommodate diverse data types from various IT systems, regardless of their structured or unstructured nature. In their design, data lakes do not impose any specific data schemas during the process of data ingestion [7]. However, due to the absence of guaranteed ACID (Atomicity, Consistency, Isolation, Durability) properties, proper monitoring, metadata management, and comprehensive quality control are crucial to prevent data lakes from transforming into what is commonly referred to as "data swamps". Modern big data frameworks like Apache Spark offer powerful tools for data transformation, validation, and enrichment at scale, contributing to the overall enhancement of data quality in data lakes.

1.1. Objectives and contribution

The main objective of this paper is to explore strategies for maintaining data quality within data lake pipelines. Additionally, we aim to assess the feasibility of using contemporary big data frameworks for this purpose. To accomplish these objectives, our study delves into data quality within data lakes, investigating the data quality-related challenges practitioners encounter during different stages of data lake pipelines. Subsequently, we examine the capabilities of modern big data frameworks, specifically Apache Spark, in executing tasks related to data quality within data lake pipelines. A thorough analysis of Spark's features is conducted, and a comparative evaluation against similar frameworks is presented. Our paper not only showcases how Spark can be efficiently leveraged to ensure data quality within ELT (Extract-Load-Transform) and data processing tasks, pipeline orchestration, and the execution of data quality tests, but also emphasizes its practical application in the agri-food domain in achieving these objectives.

The remainder of the paper is structured as follows: Section 2 presents the contributions of relevant studies regarding data quality in data lakes. In Section 3 we provide the theoretical background concerning data lakes, data quality in data lake pipelines and the Apache Spark framework. Section 4 presents the architecture model for the proposed data lake pipeline design. The results of the qualitative evaluation of using Apache Spark for maintaining data lake quality and their interpretation are presented in Section 5. Finally, we summarize our findings with possible future research directions in Section 6.

2. Related work

Data lakes, which serve as central repositories for a wide variety of raw and unstructured data, are prone to storing data of varying quality levels. Ensuring high data quality within these lakes is essential to derive accurate analyses, make informed decisions, and drive actionable insights. Nonetheless, research literature and established practices regarding data quality in data lake pipelines are somewhat limited. Generally, the challenge of maintaining data quality in data lakes arises from the diverse range of data sources and formats they accommodate [8]. These sources can range from structured databases to semi-structured and completely unstructured data like text, images, and videos. This heterogeneity often leads to issues such as inconsistencies, incompleteness, duplication, and inaccuracies within the data lake. Hence, a suitable data preprocessing and integration strategy is vital to ensure data quality and usability [9]. Furthermore, the implementation of data governance, interoperability, and standardization mechanisms is also encouraged, particularly in the context of federated data lakes [10].

Data lake practitioners propose various strategies to uphold data quality in data lakes [11], including the versioning of data to ensure data lineage, and testing which encompasses data validation, metadata testing, and continuous data integration testing. An array of tools for testing data pipelines has been developed, some of which are based on Apache Spark (e.g., Deequ). Ciaccia et al. [12] suggest the formulation and implementation of constraints to guarantee accurate and dependable data within data lakes. In this context, distinct types of constraints like schema and semantic constraints can be realized using the SPARQL language built on an RDF data model, thus preserving data integrity.

On a technical level, data lakes are usually implemented using distributed technologies such as HDFS, MapReduce, Apache Spark, etc., which provide high scalability. Sawadogo and Darmont [13] explore the complexities associated with structuring and organizing data within data lakes and emphasize the pivotal role that metadata plays in addressing these challenges. According to the authors, building a comprehensive metadata catalog serves to enforce data quality in data lakes. Mehmod et al. [9] propose a data lake pipeline architecture built on the Hadoop platform, where Apache Solr and Apache Spark are used for data analysis, and Apache Flume is used for data ingestion.

3. Preliminaries

Our proposed solution builds upon several concepts and technologies. This section provides their definition, description, and their role in the proposed data lake pipeline architecture.

3.1. Data lakes

As a data technology, data lakes appeared with the development of big data technologies from their early days. The existing data warehouses were quite strict in terms of following pre-defined data schemas, which required notable amounts of time during the integration of data collected from different data sources before storing it in the centralized data storage. Additionally, once the data was stored, there was little room to customize dashboards and queries to analyze the data besides those prepared in advance. The rapid technological development also created a paradigm shift in the data analysis approach, as business users requested more flexibility and agility in big data integration and their active inclusion in the data analysis to develop self-service business intelligence and other services [14]. As it turns out, this need was fulfilled with the introduction of data lakes.

A data lake can be defined as a "central location where it is possible to store all domain data, regardless of its source or format" [15]. Hence, it enables storing structured and unstructured data, which is an important prerequisite for numerous modern use cases. However, this does imply that a data lake architecture must be able to store large volumes of data. It also needs to include efficient data processing tools to handle data stored in its original format appropriately. A typical data lake architecture generally includes data ingestion from various data sources into the ELT pipeline, as shown in Fig. 1. As part of the ELT job, data is directly extracted and loaded in its original form into the data lake. Depending on the data analysis needs (e.g. machine learning, data analytics, business intelligence), data is then read and processed into the appropriate data format by following the "schema-on-read" approach. The ELT and schema-on-read approaches are the most common reasons for building a data lake in domains where fast data ingestion and efficient and agile processing of (un)structured data are needed.

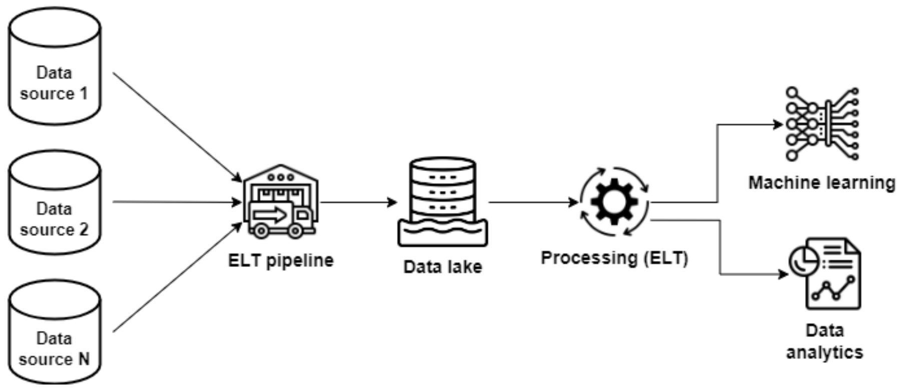


Figure 1: A general data lake architecture. Source: adjusted from [16].

3.2. Data pipeline testing

The existing body of work on data quality checking in modern data pipelines is quite limited, even more when it comes to practical examples of implementing data quality tests throughout the data pipeline. Intuitively, maintaining data quality in data pipelines demands a proper testing strategy developed upfront. A. Jain [17] presented a comprehensive list of data quality tests specifically applicable to modern data pipelines, which includes the following tests ordered in ascending order based on the required cost and time for their execution:

1. Unit test - used to validate data transformations within individual functions and queries;
2. Component test - used for schema validation of every row and column with regard to business rules;
3. Contract test - used to validate if the target table contains records present in the source table (applicable to data warehouses and data marts);

4. Flow test - used to validate the completeness, accuracy and consistency of data migrated within data flows;
5. Source test - used to ensure that all data extracted from the source systems reach the target data system without any loss or reduction;
6. Functional test - used to validate if all components within the pipeline properly work together and, when integrated, produce the expected data results;
7. Data quality matrix - used to validate and measure the accuracy, completeness, timeliness and consistency of data [18].

3.3. Big data processing frameworks

Big data processing frameworks are essential tools in modern data analysis and management, as they enable organizations to handle vast amounts of data efficiently and extract meaningful insights. These frameworks are designed to address the challenges posed by the ever-increasing volumes, velocities, and varieties of data generated in the modern digital landscape.

One of the pioneering frameworks in this field is Apache Hadoop. It introduces the concept of distributed computing, allowing data to be processed across clusters of computers in parallel. The Hadoop ecosystem comprises the Hadoop Distributed File System (HDFS) for storage and the MapReduce programming model for processing. Hadoop's scalability and fault tolerance make it suitable for handling massive datasets, making it a cornerstone for big data processing. Nevertheless, its limited support for batch processing only represents a major impediment to meeting the requirements of modern data platforms. Furthermore, its reliance on disk storage can lead to slower processing speeds, especially for iterative tasks [19].

Real-time data processing is facilitated by frameworks like Apache Flink and Apache Kafka. Apache Flink offers stream processing capabilities with event time processing, enabling organizations to analyze and act upon data as it is generated. Even though it brings some novel ideas to data processing, Flink is still considered as a less-matured framework with a more limited set of supported operations compared to Spark [19]. Apache Kafka, on the other hand, serves as a distributed streaming platform for building real-time data pipelines and applications, providing efficient data ingestion and distribution.

Apache Spark, another prominent framework, builds upon the foundations of Hadoop but offers enhanced performance and a more versatile processing model leveraging both batch and stream processing capabilities [20]. Spark employs in-memory computing, allowing iterative algorithms and interactive querying to be executed much faster compared to traditional disk-based systems [21]. It also supports various programming languages (Python, Java, Scala, and R [22]) and provides libraries for machine learning, graph processing, and stream processing, making it a comprehensive choice for diverse big data tasks. Its concise APIs and high-level abstractions make it more user-friendly, allowing developers to express complex data processing tasks in a more intuitive manner. When it comes to performance, Spark has proven to be several times faster in data access and processing than Hadoop or other disk-based approaches [23]. These results originate in Spark's programming model based on Resilient Distributed Datasets (RDDs), where objects are kept in memory to reduce the reading overhead due to disk operations between iterative computations [19]. Furthermore,

4. Data lake pipeline architecture design for the agri-food domain

This paper focuses on evaluating the suitability of Apache Spark for maintaining data quality within data lakes. To accomplish this objective, we designed a data pipeline architecture for the agri-food domain. In this context, modern data platforms supporting agri-food value chains are expected to gather and process data generated throughout various stages of the value chain, such as food production, processing, packaging, and distribution. The establishment of a data platform plays a crucial role in creating an ecosystem where data-driven policies and measures can effectively reduce food loss and waste. The potential input data for such a platform includes a range of sources, including sensor data from fields, satellite images depicting production areas, and reports provided by farmers with information about the materials and seeds used and produced. These data originate from various source systems and exhibit varying data structures, making a data lake architecture well-suited for this particular business scenario.

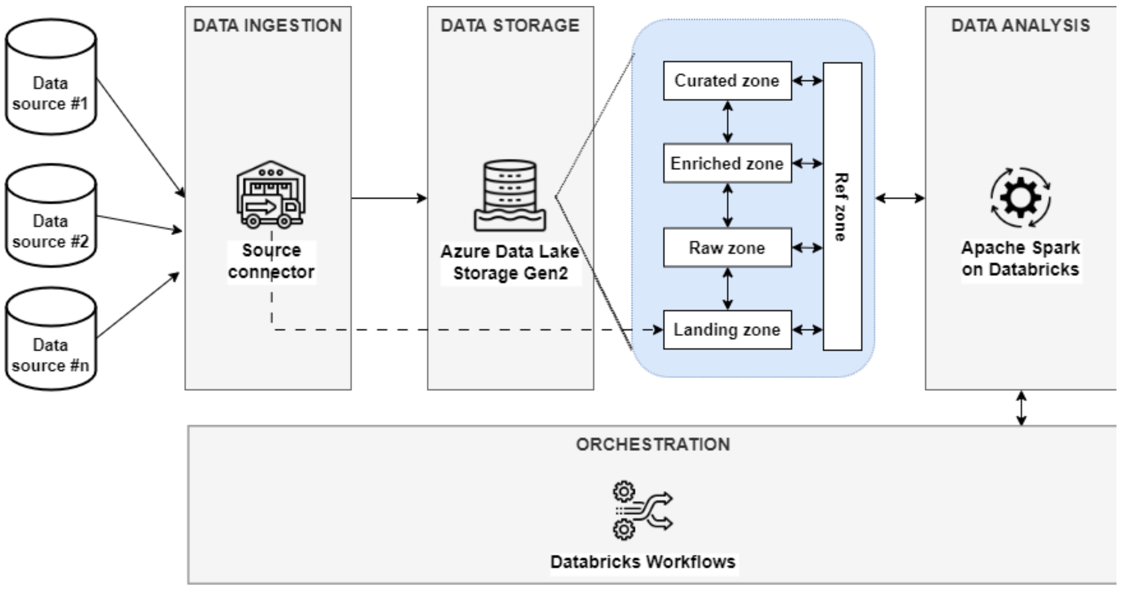


Figure 2: Data lake pipeline architecture based on Spark.

Fig. 2 presents a high-level architecture model for the data lake pipeline in our study. The data lake implementation is carried out on the Microsoft Azure platform infrastructure, which enables us to focus on pipeline implementation rather than infrastructure maintenance and overall security. The design follows a layered approach, dividing the data platform architecture into distinct layers that correspond to different stages of the data pipeline:

- Data ingestion - tools and solutions for continuously collecting raw data from source systems;
- Data storage - data systems and approaches used to store raw ingested data and/or aggregated/processed data produced within the later stage of the pipeline;

- Data analysis - tools and frameworks for processing data for analysis purposes;
- Orchestration - tools and frameworks for the automation and monitoring of pipeline stage execution.

Considering the specific characteristics of the agri-food domain's data, we have chosen to utilize the data lake as the central data repository within the data storage layer. Consequently, within the data ingestion layer, we adopt an ELT approach. To accomplish this, we developed a custom source connector designed to extract data from the source systems. Due to the unique circumstances within the agri-food domain, where data sources mainly consist of farmers and their manually submitted reports, and the source systems are typically governed by government agencies with strict access policies, we have chosen to develop a custom connector implemented as an Azure Function. This custom connector ensures regular data extraction while adhering to the necessary access policies and security requirements.

Data collected by the *Source connector* is stored in the *Azure Data Lake Storage Gen2* data lake solution, where it is divided across different zones according to their varying degree of transformation (from raw to fully processed and aggregated). Additionally, the *Ref zone* assumes the role of a data catalog, managing tasks such as assigning internal IDs, mapping these IDs to input data, and tracking the processing status of each raw data record. It serves as a reference point for maintaining data lineage and providing essential metadata for the overall data pipeline.

All these complex migrations of data between different data lake zones are handled by Apache Spark. Specifically, each migration (e.g. *Landing to Raw* migration) is handled by a dedicated Spark job specified within a Python script. Among the available Spark modules, we are using the Spark SQL module to clean and aggregate data after it lands in the data lake. The module offers several abstractions to work with data: *Dataset*, *DataFrames*, *SQL Tables* and *RDD (Resilient Distributed Dataset)*. Most of the tasks can be achieved by either of these options, but not all of the options are available for all programming languages [24]. When the execution DAG is created, all named abstractions use the same execution engine, which means they can be used interchangeably [25]. The *DataFrames* abstraction is the easiest to use, as it offers a lot of underlining optimizations for running tasks and it is available in various languages (e.g. Python, R). In essence, a *Spark DataFrame* is a distributed in-memory table-like structure with named columns and schema that defines the data type for each column [26, 24]. *DataFrames* can be created from a large array of sources spanning from structured data files, Hive tables, external databases or existing RDDs.

Finally, to reduce the system administration efforts for the Spark cluster running these jobs, we chose to deploy Apache Spark on the Azure Databricks platform, which offers a fully managed cloud service for data processing and analytics. The execution of the scripts (Spark jobs) is monitored and scheduled by the Databricks Workflows orchestration service available in the Databricks Platform.

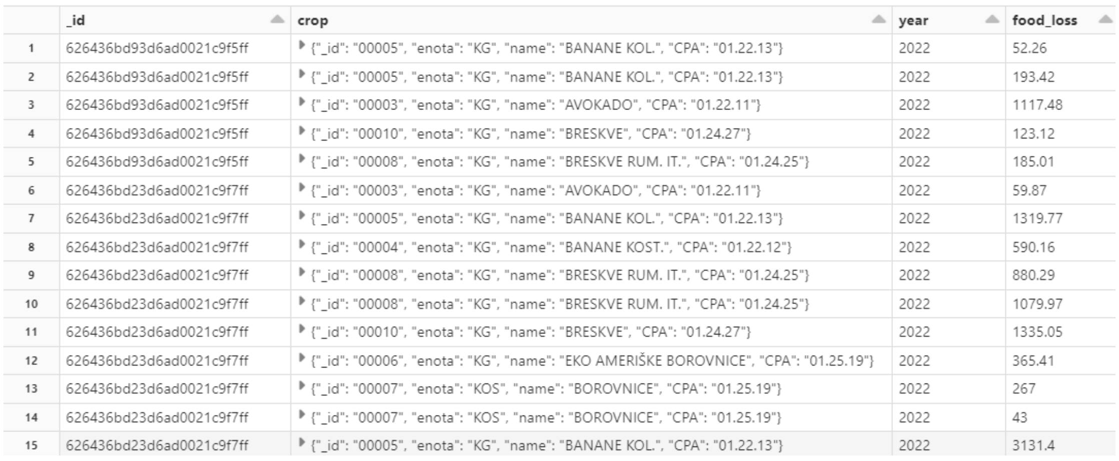
5. Results and discussion

In this section, we present the results of using Apache Spark in various stages of the data lake pipeline presented in Section 4. We discuss our empirical findings and remarks about the suitability of using Spark for maintaining data quality throughout the data processing via ELT and pipeline orchestration and testing stages.

5.1. Data processing with Spark

For clarity purposes, we will present the results of using Spark for data processing purposes only on selected segments within the ELT job pipeline.

For instance, the data processing for calculating the farmers' food loss includes processing their harvest and delivery reports to calculate the difference between the total quantity of a specific crop harvested and delivered to customers. Here, one Spark job responsible for moving data from the *Raw* to the *Enriched zone* must first process all harvest and delivery reports for each farmer stored in the *Raw zone* and group them based on the farmer's ID, crop, month and year. This result is used by another Spark notebook (job) run responsible for calculating the yearly food loss for each farmer in the *Curated zone* (Fig. 3).



	_id	crop	year	food_loss
1	626436bd93d6ad0021c9f5ff	▸ {"_id": "00005", "enota": "KG", "name": "BANANE KOL.", "CPA": "01.22.13"}	2022	52.26
2	626436bd93d6ad0021c9f5ff	▸ {"_id": "00005", "enota": "KG", "name": "BANANE KOL.", "CPA": "01.22.13"}	2022	193.42
3	626436bd93d6ad0021c9f5ff	▸ {"_id": "00003", "enota": "KG", "name": "AVOKADO", "CPA": "01.22.11"}	2022	1117.48
4	626436bd93d6ad0021c9f5ff	▸ {"_id": "00010", "enota": "KG", "name": "BRESKVE", "CPA": "01.24.27"}	2022	123.12
5	626436bd93d6ad0021c9f5ff	▸ {"_id": "00008", "enota": "KG", "name": "BRESKVE RUM. IT.", "CPA": "01.24.25"}	2022	185.01
6	626436bd23d6ad0021c9f7ff	▸ {"_id": "00003", "enota": "KG", "name": "AVOKADO", "CPA": "01.22.11"}	2022	59.87
7	626436bd23d6ad0021c9f7ff	▸ {"_id": "00005", "enota": "KG", "name": "BANANE KOL.", "CPA": "01.22.13"}	2022	1319.77
8	626436bd23d6ad0021c9f7ff	▸ {"_id": "00004", "enota": "KG", "name": "BANANE KOST.", "CPA": "01.22.12"}	2022	590.16
9	626436bd23d6ad0021c9f7ff	▸ {"_id": "00008", "enota": "KG", "name": "BRESKVE RUM. IT.", "CPA": "01.24.25"}	2022	880.29
10	626436bd23d6ad0021c9f7ff	▸ {"_id": "00008", "enota": "KG", "name": "BRESKVE RUM. IT.", "CPA": "01.24.25"}	2022	1079.97
11	626436bd23d6ad0021c9f7ff	▸ {"_id": "00010", "enota": "KG", "name": "BRESKVE", "CPA": "01.24.27"}	2022	1335.05
12	626436bd23d6ad0021c9f7ff	▸ {"_id": "00006", "enota": "KG", "name": "EKO AMERIŠKE BOROVNICE", "CPA": "01.25.19"}	2022	365.41
13	626436bd23d6ad0021c9f7ff	▸ {"_id": "00007", "enota": "KOS", "name": "BOROVNICE", "CPA": "01.25.19"}	2022	267
14	626436bd23d6ad0021c9f7ff	▸ {"_id": "00007", "enota": "KOS", "name": "BOROVNICE", "CPA": "01.25.19"}	2022	43
15	626436bd23d6ad0021c9f7ff	▸ {"_id": "00005", "enota": "KG", "name": "BANANE KOL.", "CPA": "01.22.13"}	2022	3131.4

Figure 3: An example of the resulting table with food loss information saved in the *Curated zone*.

5.2. Data lake pipeline orchestration with Spark

To schedule and execute data processing pipelines at specific times, we used Databricks Workflows. This service provides orchestration capabilities for data processing, machine learning, and analytics pipelines [27]. By using Workflows, we can create cron jobs that automate the execution of pipelines according to predefined schedules. Fig. 4 illustrates the ability to specify dependencies between notebooks responsible for specific pipeline tasks through DAGs, where multiple tasks (notebooks) can be run in parallel. For instance, in the example showcased, there is a pipeline for calculating food loss. This pipeline consists of five notebooks that handle three distinct phases of data transfer and transformation within the data lake zones: landing-raw, raw-enriched, and enriched-curated. These phases are represented horizontally in the graph, while the individual tasks are depicted vertically.

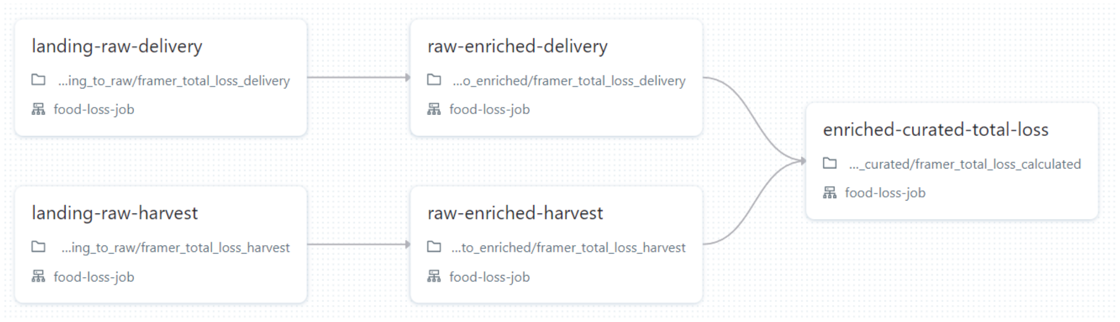

landing-raw-delivery
...ing_to_raw/framer_total_loss_delivery
food-loss-job
raw-enriched-delivery
...o_enriched/framer_total_loss_delivery
food-loss-job
enriched-curated-total-loss
..._curated/framer_total_loss_calculated
food-loss-job
landing-raw-harvest
...ing_to_raw/framer_total_loss_harvest
food-loss-job
raw-enriched-harvest
...to_enriched/framer_total_loss_harvest
food-loss-job

Figure 4: The orchestration of the food loss data pipeline in Databricks Workflows.

5.3. Data lake pipeline testing with Spark

Spark provides multiple solutions for testing purposes. When it comes to unit tests, there are several options for their implementation in Databricks. Unit tests can be implemented within the Spark notebook itself, in a separate notebook, or outside of the notebook as long as they are located within the same repository. For Python and R, the recommended approach is to store functions and their respective unit tests outside of the notebooks [28]. By doing so, tests can be executed directly from the notebook, offering the advantage of running tests easily during the development process. Alternatively, Databricks also supports running tests through the Databricks web terminal, providing an additional option for executing tests.

The example code snippet below demonstrates a unit test created for the food loss pipeline, using the Python testing library called *unittest*. Since the function being tested involves a PySpark transformation, there are various aspects that can be tested. This includes checking the correctness of the schema, data, number of rows in the DataFrame, expected values, nullability of columns, exceptions, etc. through various assertions. The provided code illustrates an example of a unit test for the same function described in Section 5.1.

```
class food_loss_test_case(unittest.TestCase):
    def test_schema(self):
        delivery_schema = StructType(
            [
                StructField("_id", StringType(), True),
                StructField(
                    "crop",
                    StructType(
                        [
                            StructField("_id", StringType(), True),
                            StructField("unit", StringType(), True),
                            StructField("name", StringType(), True),
                            StructField("CPA", StringType(), True),
                        ]
                    ),
                    True,
                ),
                StructField("year", StringType(), True),
                StructField("month", StringType(), True),
                StructField("sumDelivery", DoubleType(), True),
            ]
        )
       harvest_schema = StructType(
          [
            StructField("_id", StringType(), True),
            StructField(
                "crop",
                StructType(
                    [
                        StructField("_id", StringType(), True),
                        StructField("unit", StringType(), True),
                        StructField("name", StringType(), True),
                        StructField("CPA", StringType(), True),
                    ]
                ),
                True,
            ),
            StructField("year", StringType(), True),
            StructField("month", StringType(), True),
            StructField("sumHarvest", DoubleType(), True),
        ]
    )
    harvest_data = [
        (
            "626436bd93d6ad0321c9f7ff",
            ("00000", "KOS", "EKO ČAJ DROBNOCVETNI VRBOVEC 20G", "01.22.22"),
            "2022",
            "07",
            1492.0,
        )
    ]
    deliveries_data = [
        (
            "626436bd93d6ad0321c9f7ff",
            ("00005", "KG", "BANANE KOL.", "01.22.13"),
            "2022",
            "07",
            2142.08,
        )
    ]
    df_harvest = spark.createDataFrame(harvest_data, harvest_schema)
    df_deliveries = spark.createDataFrame(deliveries_data, delivery_schema)
    expected_schema = StructType(
        [
            StructField("_id", StringType(), True),
            StructField(
                "crop",
                StructType(
                    [
                        StructField("_id", StringType(), True),
                        StructField("unit", StringType(), True),
                        StructField("name", StringType(), True),
                        StructField("CPA", StringType(), True),
                    ]
                ),
                True,
            ),
            StructField("year", StringType(), True),
            StructField("food_loss", DoubleType(), True),
        ]
    )
    df_food_loss = self.calculate_food_loss(df_deliveries, df_harvest)
    self.assertEqual(df_food_loss.schema, expected_schema)
```

6. Conclusion

To summarize, the exponential growth of data in modern IT systems requires the implementation of robust data quality control mechanisms to ensure the reliability and accuracy of business decisions. This becomes even more crucial in big data systems, where unique data quality challenges arise across the entire data processing pipeline. The focus of this paper was to address the importance of data quality in data lakes, which often lack adequate mechanisms, resulting in data swamps and inaccurate outcomes. We studied the capabilities of the Apache

Spark framework and, particularly, its features that can be used to implement data lake pipelines while simultaneously ensuring data quality. The paper showcased a specific agri-food use case where Spark was successfully employed in various stages of the data pipeline to maintain the quality of data throughout the pipeline. Specifically, we empirically evaluated Spark's suitability for data processing and pipeline orchestration and testing purposes. Overall, our findings confirm the suitability of Apache Spark for addressing data quality concerns in data lake architectures. As part of our future work, we plan to run a performance analysis of our proposed data lake pipeline architecture and develop a comprehensive data lake quality framework, which will include suitable mechanisms to measure relevant data quality dimensions.

Acknowledgments

This work was supported by the Slovenian Research Agency (Research Core Funding) under Grant P2-0057, and also in part by the European Union's Horizon 2020 Research and Innovation Programme (Grant Agreements No. 1001036388 and 101059473).

References

[1] M. Talha, A. Abou El Kalam, N. Elmarzouqi, Big data: Trade-off between data quality and data security, Procedia Computer Science 151 (2019) 916–922.

[2] ISO 25012:2008(E), Software Engineering - Software Product Quality Requirements And Evaluation (SQuaRE) - Data Quality Model, Standard, International Organization for Standardization, Geneva, CH, 2008.

[3] P. Zhang, F. Xiong, J. Gao, J. Wang, Data quality in big data processing: Issues, solutions and open problems, in: 2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), IEEE, 2017, pp. 1–7.

[4] A. Ramasamy, S. Chowdhury, Big data quality dimensions: a systematic literature review, JISTEM-Journal of Information Systems and Technology Management 17 (2020).

[5] M. Abdallah, M. Muhairat, A. Althunibat, A. Abdalla, Big data quality factors, frameworks and challenges, Compusoft 9 (2020) 3785–3790.

[6] L. Cai, Y. Zhu, The challenges of data quality and data quality assessment in the big data era, Data science journal 14 (2015).

[7] C. Mathis, Data lakes, Datenbank-Spektrum 17 (2017) 289–293.

[8] C. Giebler, C. Gröger, E. Hoos, H. Schwarz, B. Mitschang, Leveraging the data lake: Current state and challenges, in: Big Data Analytics and Knowledge Discovery: 21st International Conference, DaWaK 2019, Linz, Austria, August 26–29, 2019, Proceedings 21, Springer, 2019, pp. 179–188.

[9] H. Mehmood, E. Gilman, M. Cortes, P. Kostakos, A. Byrne, K. Valta, S. Tekes, J. Riekki, Implementing big data lake for heterogeneous data sources, in: 2019 ieee 35th international conference on data engineering workshops (icdew), IEEE, 2019, pp. 37–44.

[10] J. Eder, V. A. Shekhovtsov, Data quality for federated medical data lakes, International Journal of Web Information Systems 17 (2021) 407–426.

[11] T. lakeFS team, How To Maintain Data Quality In Your Data Lake, 2023. URL: https://lakefs.io/blog/how-to-maintain-data-quality-in-your-data-lake/.

[12] P. Ciaccia, D. Martinenghi, R. Torlone, et al., Conceptual constraints for data quality in data lakes, in: CEUR WORKSHOP PROCEEDINGS, volume 3340, CEUR-WS, 2022, pp. 111–122.

[13] P. Sawadogo, J. Darmont, On data lake architectures and metadata management, Journal of Intelligent Information Systems 56 (2021) 97–120.

[14] A. Gorelik, The enterprise big data lake: Delivering the promise of big data and data science, O'Reilly Media, 2019.

[15] A. LaPlante, Architecting data lakes, O'Reilly Media, 2016.

[16] M. Šestak, M. Turkanović, Pregled in analiza tehnoloških skladov za implementacijo sodobnih it arhitektur velepodatkov, Uporabna informatika 31 (2023).

[17] A. Jain, Everything you need to know about testing data pipelines, 2023. URL: https://www.thoughtworks.com/insights/blog/testing/testing-data-pipelines.

[18] V. Duraisamy, ETL testing — How to test your data pipelines the right way, 2023. URL: https://towardsdatascience.com/forget-about-the-new-data-trends-in-2023-d2756add3317.

[19] J. Veiga, R. R. Expósito, X. C. Pardo, G. L. Taboada, J. Tourifio, Performance evaluation of big data frameworks for large-scale data analytics, in: 2016 IEEE International Conference on Big Data (Big Data), IEEE, 2016, pp. 424–431.

[20] E. Shaikh, I. Mohiuddin, Y. Alufaisan, I. Nahvi, Apache spark: A big data processing engine, in: 2019 2nd IEEE Middle East and North Africa COMMunications Conference (MENACOMM), IEEE, 2019, pp. 1–6.

[21] D. García-Gil, S. Ramírez-Gallego, S. García, F. Herrera, A comparison on scalability for batch big data processing on apache spark and apache flink, Big Data Analytics 2 (2017) 1–11.

[22] H. Karau, A. Konwinski, P. Wendell, M. Zaharia, Learning spark: lightning-fast big data analysis, " O'Reilly Media, Inc.", 2015.

[23] V. S. Jonnalagadda, P. Srikanth, K. Thumati, S. H. Nallamala, K. Dist, A review study of apache spark in big data processing, International Journal of Computer Science Trends and Technology (IJCST) 4 (2016) 93–98.

[24] B. Chambers, M. Zaharia, Spark - the definitive guide, O'Reilly Media, 2018.

[25] Spark SQL, DataFrames and datasets guide, n.d. URL: https://spark.apache.org/docs/latest/sql-programming-guide.html.

[26] J. Damji, B. Wenig, T. Das, D. Lee, Learning Spark, 2 ed., O'Reilly Media, Sebastopol, CA, 2020.

[27] What is Databricks Workflows?, n.d. URL: https://docs.databricks.com/workflows/index.html.

[28] Unit testing for Notebooks, n.d. URL: https://docs.databricks.com/notebooks/testing.html.