Politecnico di Torino

Politecnico di Torino

Master's Degree in Computer Engineering
A.a. 2025/2026

Driving Business Value with Data

A Data Engineering Case Study at L'Oréal

Supervisor:
Paolo Garza

Candidate:
Alessio Marino

Abstract

In the contemporary digital economy, the exponential growth of data presents both opportunities and challenges for organizations. The primary challenge lies in collecting data and in transforming it into actionable information that can effectively guide strategic decisions.
This is the case for L'Oréal Group, the world's largest cosmetics group, which has decided to put data at the centre to sustain its competitive advantage, and become a data-driven enterprise, making informed choices through business intelligence and with the help of machine learning and other cutting-edge technologies.
This research underscores the critical contribution of Google Cloud services in facilitating this transformation, notably in guiding the shift from on-premises infrastructures toward distributed cloud architectures.
Furthermore, it investigates critical stages of data engineering, including the design and implementation of data visualization solutions, to convert complex datasets into insightful reports and to optimize operational costs. It addresses issues of data governance, with emphasis on database management, security, and interoperability, as essential components of a reliable data ecosystem. More broadly, the application of machine learning to automate repetitive classification tasks and streamlining workflows to enable greater attention to activities of higher strategic importance.
This analysis illustrates the convergence of data engineering, cloud computing, and machine learning in delivering scalable and efficient solutions that foster innovation within a global enterprise.

Table of Contents

Chapter 1

Introduction

In the contemporary business landscape, data has emerged as one of the most valuable assets for organizations across industries. Its effective use plays a central role in securing and maintaining competitive advantage.
The true value of data lies in its transformation into information that can guide strategies. The ability to collect, manage, and analyze increasingly vast datasets has therefore become a defining capability for global leaders, such as L'Oréal, where data-driven insights are indispensable in shaping consumer understanding, refining marketing strategies, and fostering innovation in product development. L'Oréal is a highly influential cosmetics group, both in France and globally, which provided an enriching environment for me to grow as a data professional.

The present thesis investigates the crucial role of data engineering in enabling L'Oréal—and by extension, other data-centric organizations—to harness their extensive data resources effectively. Data engineering encompasses the design, construction, and management of systems and pipelines that convert raw data into usable formats. In particular, I was entrusted with testing Google Cloud Platform (GCP) tools, a suite of modular cloud computing services including data storage, data analysis and machine learning, as well as a range of management tools.
Previously, companies with servers on their premises had to manage the infrastructure. Today, the infrastructure is managed directly by cloud providers. We call this infrastructure as a service (IaaS). Leading the market is AWS, Amazon's cloud service, launched in March 2006 with the aim of revolutionising and helping developers in the provision of servers and storage space. The main components available in early 2006 were on-demand virtual machines and persistent storage space. In the following years, other cloud providers, such as Google with Google Cloud Platform, began offering this innovative service, such as Google with Google Cloud Platform in April 2008 and Microsoft Azure in 2010. Other cloud providers such as Alibaba Cloud and Oracle Cloud also share the market, but with smaller market shares. The initial offering has continued to evolve, with different cloud

providers offering services such as machine learning with very high computing capabilities that can be executed with almost no programming knowledge. Other services, which are increasingly innovative, are still emerging today with the aim of assisting with programming and migrating tools to the cloud.

The growing reliance on Big Data in decision-making highlights both opportunities and challenges. The enormous volumes of structured and unstructured data generated through mobile devices, multimedia, and digital transactions demand sophisticated pre-processing before they can be rendered suitable for analysis. Advances in cloud computing have significantly supported this evolution, offering the computational power, networking capabilities, and storage capacity required for managing large-scale data workloads. Cloud-based infrastructures enable parallel processing, scalability, virtualization, and enhanced security, while simultaneously reducing the costs traditionally associated with hardware and physical data centers.

1.1 Goal

During the final year of a Master's program in Data and Artificial Intelligence in Paris, France, I undertook advanced coursework in subjects such as Big Data, Database Management, Data Processing, Cloud Virtualization, Data Visualization, and Machine Learning. This theoretical foundation provided the basis for applying knowledge in a professional context. In the second semester, I had the opportunity to complete a six-month internship at L'Oréal Group, from March to August 2023, at the Levallois-Perret headquarters, which oversees the company's operations across Europe, the Middle East, and Africa (EMEA).
As a Data Engineer within the organization, the primary responsibilities involved collaborating with colleagues to develop data-driven use cases, participating in agile workflows, and contributing to the design and implementation of data pipelines. These activities required overseeing the entire lifecycle of data treatment, including the retrieval of information from diverse sources, ensuring its accessibility across different business domains, and constructing dashboards. Over time, this role enabled me to gain autonomy in developing use cases end-to-end, from initial conception through deployment.

This thesis provides a comprehensive account of the projects completed during the internship, with a particular focus on three initiatives, which represent the pillars of big business: data visualisation, data gouvernance and machine learning. Each of these projects will be examined in detail, with attention given to their objectives, methodologies, challenges, and outcomes.

1.2 Thesis Structure

This thesis is organized into three main chapters, each of which contributes to a comprehensive understanding of the research context.

The first chapter provides an overview of L'Oréal Group, beginning with a description of the company's history, global significance, and position within the cosmetics industry. Particular attention is devoted to its commitment to innovation and its increasing reliance on data-driven strategies.
This chapter then introduces the Data Engineering department at L'Oréal, examining its fundamental role in enabling the company's digital transformation. To situate this within a broader perspective, the chapter explores the evolution of Big Data, tracing its development from on-premises systems to cloud-based platforms, and discusses the impact of distributed data processing brought about by the growth of the internet. It also considers the motivations for and benefits of cloud migration, highlighting how these technological shifts have redefined the role of the data engineer in modern enterprises. Finally, the chapter introduces the use of machine learning in corporate contexts, before reviewing key technologies and tools employed within data engineering, such as infrastructure-as-a-service, continuous integration and delivery practices, data ingestion, data quality management, and visualization.

The second chapter of this thesis is devoted to the methodology, outlining the research approach. It presents the three principal projects carried out during the internship, describing their objectives, design, implementation, and outcomes.

- The first project concerns the development of a dashboard designed to support data analysis and reporting activities. The system architecture integrated Looker Studio, BigQuery, and Terraform, creating a robust environment for data visualization and infrastructure management. The project began with the extraction of logs and metadata, which were subsequently processed and structured for visualization. The implementation was carried out with an emphasis on industrialisation through infrastructure as code, thereby ensuring scalability, reproducibility, and maintainability. This project was particularly significant as it constituted the foundation of my practical experience in data engineering, providing insight into the operational responsibilities and technical expertise required in the role.

- The second project involved the design and implementation of a proof of concept for Dataplex, with the aim of enhancing L'Oréal's data management capabilities. This initiative introduced broader concepts such as data lakes and data mesh, situating Dataplex within the context of modern data architecture. The objectives of the project were clearly defined, and the system architecture was carefully mapped out to demonstrate how Dataplex could be integrated

into existing workflows. During the development process, several challenges were identified, particularly in adapting the tool to L'Oréal's data landscape. These were addressed through iterative problem-solving strategies, and the project concluded with a successful demonstration of Dataplex's potential contributions to the company's data infrastructure.

- The final project focused on the implementation of the project CrunchML using Vertex AI in conjunction with BigQuery ML.
  The primary objective of this initiative was to leverage machine learning for data classification tasks. The first phase of the project concentrated on model development, where different approaches were tested to achieve an acceptable level of accuracy in predicting a target column from a given dataset. Once a satisfactory model was obtained, the focus shifted to the creation of a comprehensive infrastructure and workflow. This advanced stage of the project incorporated multiple cloud-based tools and services, including Cloud Run, Cloud Functions, BigQuery ML, and Docker, which together enabled the deployment, scaling, and automation of the classification system.

The third chapter offers an analysis of results of the three projects. The impact of the solutions implemented in each project is evaluated, considering their effectiveness and contribution to L'Oréal's overall data strategy. A comparison is made between the actual results and the initial expectations set out at the beginning of each project. The chapter concludes with a discussion of the lessons learned throughout the projects, highlighting both successes and areas for potential improvement.

The thesis concludes by summarizing the key findings from the research, emphasizing the practical and theoretical implications of the work. It acknowledges the limitations of the study, discussing any constraints or challenges encountered during the research process. It ends with some perspectives for future research or projects.

Chapter 2

Corporate Context and Technologies

L'Oréal Group is the largest cosmetics group worldwide and known for its extensive range of beauty products and innovative research in the field of cosmetics. Founded in 1909 in France by Eugène Schueller, it has grown to become a global leader in the beauty industry, and today it boasts an impressive portfolio of 36 brands [1], the best known are Lancôme, Prada, Garnier, Ralph Lauren, Yves Saint Laurent, Armani, La Roche Posay and so on.
Headquartered in Clichy, Hauts-de-Seine, in France, with 88 thousand dedicated employees in 150 countries in 2023, L'Oréal continues to drive innovation and set trends in the cosmetics market. As a result, the company achieved remarkable success in 2019, generating total sales of 29.9 billion euros.
Additionally, it aims to provide a diverse and comprehensive range of brands to meet the needs and desires of consumers worldwide: perfect blend between European, American, Chinese, Japanese, Korean, Brazilian, Indian and African brands.

The core orientation lies in innovation, research, and product development, in fact, the company invests heavily in Research and Development (R&D) to introduce new and advanced beauty products.
L'Oréal's commitment to social and environmental issues is deeply rooted in its strategy, offering numerous benefits that align with its long-term goals. The company's focus on sustainability not only enhances its reputation and brand image but also provides a competitive edge by attracting socially and environmentally conscious consumers. To demonstrate its dedication, L'Oréal has launched several initiatives, for example the company prioritizes sustainable sourcing, ensuring that raw materials are ethically obtained with minimal environmental impact.
L'Oréal has also made significant strides in reducing its carbon footprint, achieving a 91% reduction in emissions from its industrial sites since 2005. By the end of

2025, the company aims to achieve carbon neutrality across all sites, with plans to further reduce greenhouse gas emissions by 2030 [2].

L'Oréal also champions diversity and inclusion within its workforce and the broader beauty industry. They actively promotes gender equality and provides support for employees with disabilities. The L'Oréal Foundation's partnership with UNESCO highlights this commitment, empowering women in science and fostering the next generation of female researchers [3]. The group engages in community empowerment through initiatives like Citizen Day, where employees worldwide volunteer to support social and environmental causes.

In the past three to five years, customer expectations in the beauty industry have undergone significant transformations, surpassing the changes observed in the previous three decades. Customers now seek richer experiences and more comprehensive services and advice.
To meet these evolving aspirations, the company has harnessed the power of technology to create revolutionary Beauty Experiences. Their approach revolves around the integration of digital solutions, leveraging the advancements brought about by Beauty Tech. This redefines their relationship with customers by offering online skin diagnostics, expediting the launch of new products in the market, and enabling voice-activated purchases.
All these initiatives aim to provide customers with a seamless and exceptional experience, both online and in-store, in line with their expectations. With the ambition to become the leading Beauty Tech company, L'Oréal's transformation program experienced significant acceleration in 2020. To boost innovation and strengthen business partnerships, the Tech Accelerator program was born. This initiative includes expertise centers in Paris and Shanghai and, leveraging data science, user experience design, platform knowledge, and technological engineering, these centers create real-time, scalable solutions with enhanced agility.
By harnessing the power of data, L'Oréal aims to anticipate market demands, and deliver products that resonate with consumers on a global scale. This unwavering dedication to leveraging data-driven insights drives their journey toward excellence and leadership in the world of Beauty Tech.
For instance, At Viva Technology 2024, an annual event dedicated to technological innovation and start-ups, at Paris Expo Porte de Versailles, L'Oréal has introduced several ground-breaking innovations under the theme 'Beauty for Each, Powered by Beauty Tech'. Key highlights include advanced beauty devices and cutting-edge technologies designed to enhance personalization and inclusivity in beauty.
Among the new offerings, Kiehl's Derma-Reader employs advanced imaging technology to analyse skin and provide tailored skincare recommendations. Lancôme utilizes nanochip technology to boost the effectiveness of skincare products. L'Oréal Paris introduced a GenAI-powered assistant that delivers personalized beauty advice, while the L'Oréal Professionnel uses an infrared hair dryer, which promises

improved hair quality and faster drying. The latest CreAITech Lab represents L'Oréal's commitment to innovation in creativity, using GenAI to produce brand-compliant content and explore new creative technologies. This initiative includes a partnership with Meta and collaboration with a team of 30 creators specializing in 3D, AR, and AI.
A significant highlight is the Skin Technology by L'Oréal, a bio-printed skin model that mimics real human skin conditions such as eczema and acne. This advanced technology aims to enhance scientific research and support cruelty-free product testing. In terms of sustainability, the group is focused on measuring and reducing CO2 emissions from digital activities. This effort is supported through partnerships with IMPACT+, ADGREEN, and FRUGGR, which help measure the carbon footprint of digital media, content production, and website operations.

2.1 Moving to a Data-Driven Company

Evidently, the company's desire to use new technologies is well in place, which is why from 2019, the group invested in the creation of a new Data department. This approach qualifies them to understand customer behaviour, preferences, and market trends better, leading to more effective product development, marketing strategies, and operational efficiency. The processed data belongs to the eighteen different domains, as shown in the figure below and each department accesses the data it needs. The subject of this document will be within the scope of the domains of Product, Supply Chain and Finance Domain, and potentially with sell-out data.

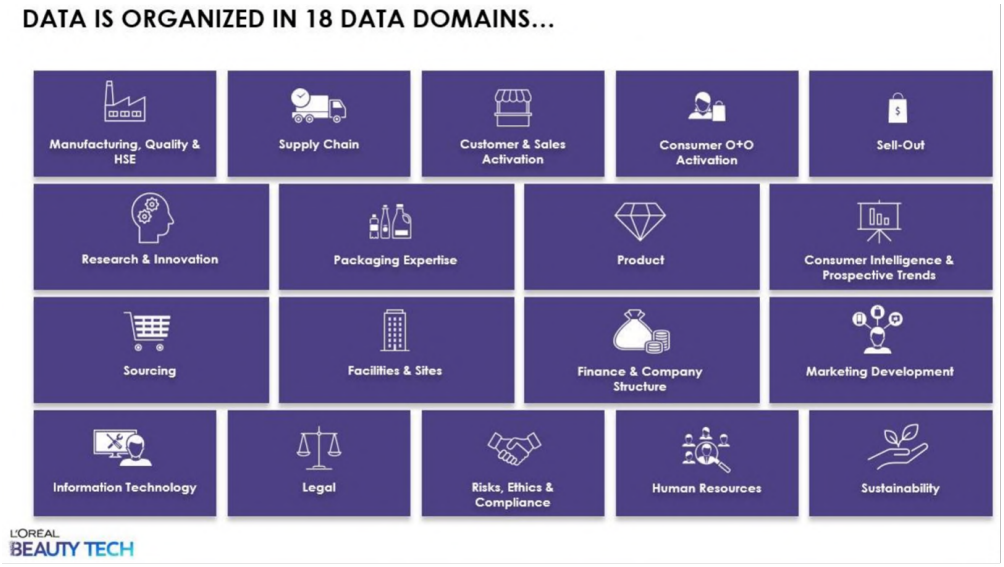


Figure 2.1: All the data domains of L'Oréal Group

In the Data Factory department, the hierarchical structure begins with the head of Data Engineers, the manager who supervised my internship, who reports directly to the IT Director for Europe. The head of Data Engineers coordinates closely with leaders in other key areas, including the Delivery Manager, Data Architect, Project Management Officer (PMO), and Data Security Manager.
The department also includes leadership roles such as the IT Europe GCP Platform Lead, the Data Visualization Manager in Poland, and the overseer of data service delivery in the United Kingdom. Additionally, two GCP Data Tech Leads are instrumental in providing technical leadership for GCP initiatives.
The department's efforts are further supported by a substantial team of external Data Engineers, who serve as consultants based in Spain and France.

The adopted methodology was "Agile", a framework designed to manage projects in a flexible and adaptive way. Agile is rooted in the idea that requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams. Instead of relying on rigid planning and long development cycles, Agile emphasizes iteration, where work is organized in short cycles, called sprints, that allow continuous feedback, adaptation, and improvement. This approach creates a dynamic process in which tasks can be adjusted according to changing needs and priorities, ensuring that the final outcome is more aligned with the expectations of stakeholders.
Central to Agile is constant communication, both within the team and with external collaborators, which makes it possible to quickly identify challenges and address them in a timely manner. With flexibility, Agile fosters a workflow that is both productive and responsive to change, making it particularly suitable for complex projects where conditions are not always predictable.

The company's motto makes clear how central data is to its vision and strategy. By emphasizing a future powered by data, technology, and artificial intelligence, the company positions itself at the intersection of science and creativity, where consumer expectations are met with tailored and precise solutions. *"We are shaping the future of beauty, pioneering Beauty Tech and leading marketing in the digital age with science, technology and creativity. A future powered by data, tech and artificial intelligence that creates new relationships with our consumers, with more inclusivity, accessibility and precision, answering their unmet needs and aspirations for beauty."*

In this sense, data is a means of deepening consumer connection, accompanying evolution of beauty, which is both technologically advanced and human-centered.

That said, it is essential to quickly delve into the origin of this data. The exploration of data origin will help to identify the nature of the specific projects and areas where the employs are involved every day.

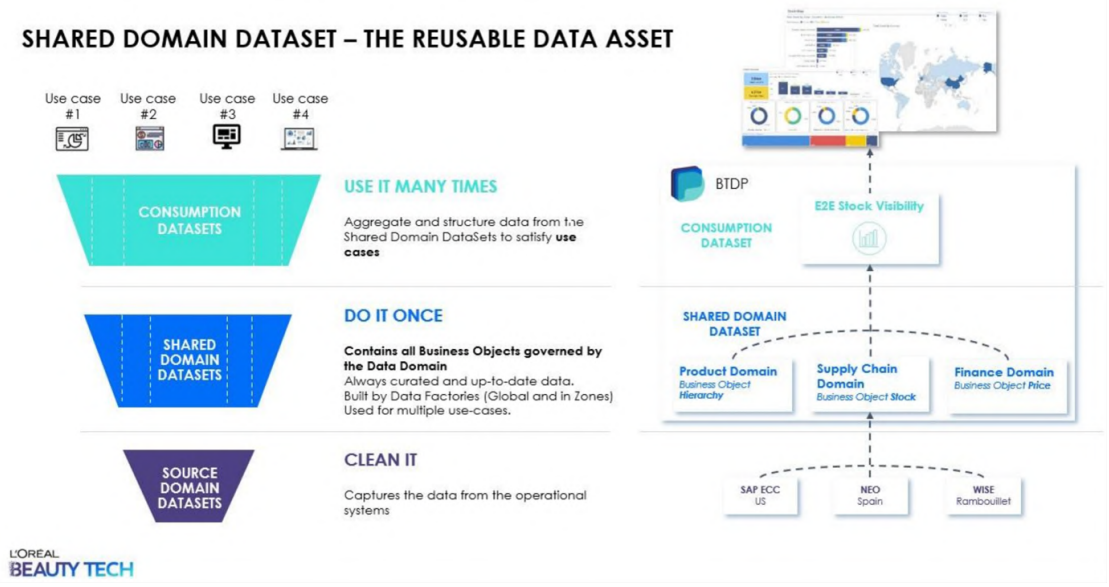


Figure 2.2: Shared Domain Dataset

Looking from the bottom up the depicted image above, this inverted pyramid shows that the data alimenting the ecosystem are diverse, although the most common source is SAP. These data streams are managed, cleaned, and consolidated by dedicated groups called Data Factories (both in Global and Zone scope) responsible for the Shared Domain Dataset (SDDS), which serves as the foundational data repository. The focused area of the Data Factory team lies here and in the upper segment, where consumption datasets are situated. Specifically, my scope centres around the European region, known as EMEA (Europe, Middle East, and Africa), where I supported the use cases originating from the specific requirements and demands of the countries within this geographical area.

We have mentioned the cloud as a valuable resource for data-driven companies. However, to understand why it is used, it is perhaps helpful to look at the history of big data and how it has evolved.

2.2 The Evolution of Big Data: From On-Premises Systems to Cloud Platforms

The origins of data management can be traced back to the development of relational database systems in the 1970s, such as IBM System R and Oracle. These systems, based on the relational model and SQL, provided a structured way to store and query enterprise data. For several decades, relational databases and traditional

data warehouses, dominated the field, supporting analytical workloads and decision-making processes within organizations.
However, with the rise of the internet and digitalization in the 1990s and early 2000s, data volumes began to grow exponentially. Traditional systems, which were optimized for structured data and predictable workloads, struggled to cope with the volume, velocity, and variety of new data sources — a challenge later described as the "3Vs" of Big Data [4]. Volume refers to the massive amount of data being generated and collected from various sources like social media, IoT devices, and digital transactions. Handling this immense quantity of data involves significant storage, processing, and computational resources. Velocity deals with the speed at which data is produced and needs to be processed. With data flowing in real-time or near-real-time, businesses must quickly analyse it to make timely decisions. Variety involves the different types and formats of data, including structured, semi-structured, and unstructured forms like text, images, and videos. This diversity makes it challenging to integrate and analyse the data effectively. Addressing these challenges requires advanced tools and scalable solutions to turn Big Data into actionable insights.

In modern infrastructure, we can distinguish between two different databases: relational and non-relational. The choice between them is rarely a trivial decision, as it carries significant implications for performance, scalability, and flexibility. To fully appreciate their roles, it is necessary to begin with the nature of data itself and the ways in which it is processed and consumed by businesses and organizations.
Data in contemporary systems is often divided into operational and analytical categories. Operational data concerns the immediate, day-to-day functioning of businesses, encompassing tasks such as inventory tracking, banking transactions, or real-time order processing. Because it must remain fresh and accurate, operational data is handled primarily by Online Transaction Processing (OLTP) systems.
Analytical data, by contrast, focuses on long-term insights: it is collected over extended periods and mined to reveal customer behavior, predict trends, or evaluate product performance. This type of data is processed through Online Analytical Processing (OLAP) systems, data warehouses, and data lakes.
Relational databases, also known as Relational Database Management Systems (RDBMS), have historically dominated this space. Their model is based on organizing data into tables consisting of rows and columns, where each row represents a unique record and each column represents a particular attribute of that record. The relational aspect derives from the use of primary keys, which uniquely identify rows in a table, and foreign keys, which create links between tables. By linking related information across multiple tables, relational databases allow for structured, well-organized storage and retrieval. The Structured Query Language, or SQL, is the primary means of interacting with these systems, enabling developers and analysts alike to perform operations ranging from simple queries to complex joins

and aggregations.
To illustrate, consider the operation of an online store. Customer information, order details, and product specifications would each reside in separate tables. A customer's order record would reference product identifiers, which in turn point to product details stored elsewhere. By joining these tables, one can reconstruct the full picture of a purchase.
This form of normalized design ensures both clarity and efficiency in storage. Relational databases further guarantee reliability through ACID compliance, an acronym for atomicity, consistency, isolation, and durability. These principles safeguard the integrity of transactions by ensuring that they either succeed in their entirety or leave the database unchanged, a property particularly vital in domains such as banking where partial updates could have severe consequences. By enforcing primary and foreign keys, relational databases minimize duplication and maintain referential integrity. Normalization processes reduce data anomalies and conserve storage space, making them cost-effective in structured scenarios. SQL's declarative nature, designed to resemble human language, further contributes to accessibility, allowing even non-specialists to query data with relative ease [5].
Despite these strengths, relational databases are not without limitations. Their design was historically intended for deployment on single machines, making vertical scaling—the addition of more powerful hardware—the default path for growth. This approach is inherently constrained, both economically and technologically, as costs escalate and physical limits are eventually reached. Moreover, the rigid schema, while helpful for consistency, often becomes a liability when requirements evolve. Altering data structures necessitates downtime and complex migrations, which may hinder innovation or responsiveness in dynamic environments. Performance, too, can degrade as data volume increases, especially when queries involve multiple joins across large tables.
It is in addressing these limitations that non-relational databases, or NoSQL systems, find their strength. Unlike relational systems, non-relational databases eschew the rigid tabular model in favor of more flexible paradigms tailored to diverse data types and workloads. Their architecture is often designed with the cloud in mind, emphasizing horizontal scaling through the distribution of data across multiple nodes. This allows them to leverage clusters of commodity hardware, rather than relying on a single powerful machine.

Non-relational databases are not monolithic and encompass several distinct models. Document databases, such as MongoDB, store information in JSON-like documents capable of representing nested structures, arrays, and varied data types. In the context of the online store, rather than dispersing information across multiple tables, a document database could encapsulate all customer details—including orders and addresses—within a single document. Moreover, document databases are well-suited to emerging fields such as artificial intelligence, where embeddings

and vector representations can be stored alongside conventional data, facilitating hybrid search capabilities that blend vector and textual queries.
Key-value stores represent another form of non-relational database, relying on the simplicity of unique keys paired with arbitrary values. Their minimalistic design ensures rapid read and write performance but restricts their applicability to simpler use cases. Graph databases, by contrast, are highly specialized for data where relationships themselves are central. Using nodes and edges, they excel at queries concerning connectivity and relationships, such as social networks or recommendation systems. Wide-column databases present yet another model, resembling relational systems superficially but allowing columns to vary across rows and distribute across servers, making them adept at managing large-scale, unstructured, or semi-structured datasets. The advantages of non-relational databases extend beyond flexibility of schema. Because documents or key-value pairs are often self-contained, a single query may retrieve all necessary data without the overhead of complex joins. Their architecture is inherently amenable to sharding, the process of partitioning data across multiple servers, ensuring that growth can be managed horizontally. They are also generally better positioned to handle unstructured or semi-structured data, which increasingly dominates modern applications in domains ranging from web platforms to IoT and AI systems. Nevertheless, non-relational systems also carry trade-offs. Their schema flexibility can lead to inconsistency if not carefully managed. Some models, such as key-value stores, are too simplistic for complex relational queries, while others, such as graph databases, lack standardized query languages, complicating interoperability. Wide-column databases, while powerful in big data contexts, often falter when transactional integrity is paramount. Thus, the choice between relational and non-relational systems is not a matter of superiority but of suitability to the task at hand.
Relational databases remain the preferred option when data is predictable in structure and when relationships between entities are both complex and essential. They are invaluable in scenarios where normalization reduces redundancy and where decades of accumulated tools and expertise can be leveraged. Non-relational databases, on the other hand, excel when flexibility, scalability, and the ability to handle unstructured or evolving data are prioritized. They are particularly well-suited to cloud environments, big data analytics, and applications requiring high performance and availability under variable loads.
Generally, many organizations find themselves employing both paradigms, integrating OLTP and OLAP systems that draw upon the strengths of each. Tools now exist, such as intelligent migration utilities, that facilitate movement between relational and document-oriented systems.

The early 2000s saw the emergence of distributed processing frameworks. A pivotal milestone was the release of Apache Hadoop in 2006, inspired by Google's papers on the Google File System (GFS) and MapReduce [6]. Hadoop introduced

a way to store massive datasets across clusters of commodity hardware and process them in parallel, drastically reducing the cost of managing large-scale data. Alongside Hadoop, a new generation of NoSQL databases such as HBase, Cassandra, and MongoDB emerged, designed to handle unstructured and semi-structured data more flexibly.

However, despite its success, Hadoop's batch-oriented model was insufficient for real-time needs. This limitation drove the development of more advanced platforms such as Apache Spark, first released in 2009 at UC Berkeley's AMPLab and later becoming an Apache project in 2013.
Spark enabled in-memory computation, real-time streaming, and machine learning capabilities, positioning itself as a more versatile alternative to MapReduce. Additional ecosystem tools like Hive and Impala further enhanced query performance and interactive analytics on top of Hadoop clusters.

In the early Big Data era, organizations typically deployed and managed these systems on-premises, which required significant investment in infrastructure, hardware, and skilled personnel for cluster configuration, monitoring, and scaling. While effective, these setups were complex, expensive, and difficult to maintain at enterprise scale. So, the advent of cloud computing marked a paradigm shift. Initially, enterprises migrated their on-premises clusters to cloud-based Infrastructure-as-a-Service (IaaS), essentially replicating existing architectures. However, cloud providers soon introduced fully managed, serverless solutions such as Amazon Redshift, Google BigQuery, and Azure Data Lake, which abstracted away operational complexity and allowed organizations to focus directly on data analysis and insights. The pay-per-use model of cloud resources also made large-scale analytics more accessible, as costs were tied to actual consumption rather than fixed hardware investments. Today, Big Data and cloud technologies are deeply intertwined. Research highlights how the cloud has enhanced the scalability, elasticity, and efficiency of Big Data solutions, particularly in the context of IoT and real-time data streams.

2.3 Foundations of Distributed Data Processing

The emergence of Apache Hadoop marked a decisive moment in the evolution of large-scale data processing. When in 2000s, organizations began to confront the reality that datasets had grown beyond the capacity of single servers, both in size and in complexity, Hadoop offered an open-source framework that redefined how organizations could store and analyse data by distributing both computation and storage across clusters of commodity hardware. At the heart of Hadoop lies the Hadoop Distributed File System (HDFS), a storage system that fragments files into large blocks and distributes them across nodes in a cluster. This design

ensured resilience through replication, with multiple copies of each block stored on different machines to safeguard against inevitable hardware failures. A central master node, the NameNode, maintained metadata about file locations, while DataNodes managed the actual storage of blocks. This architecture provided both scalability and fault tolerance, two characteristics that were previously prohibitively expensive in proprietary storage systems.

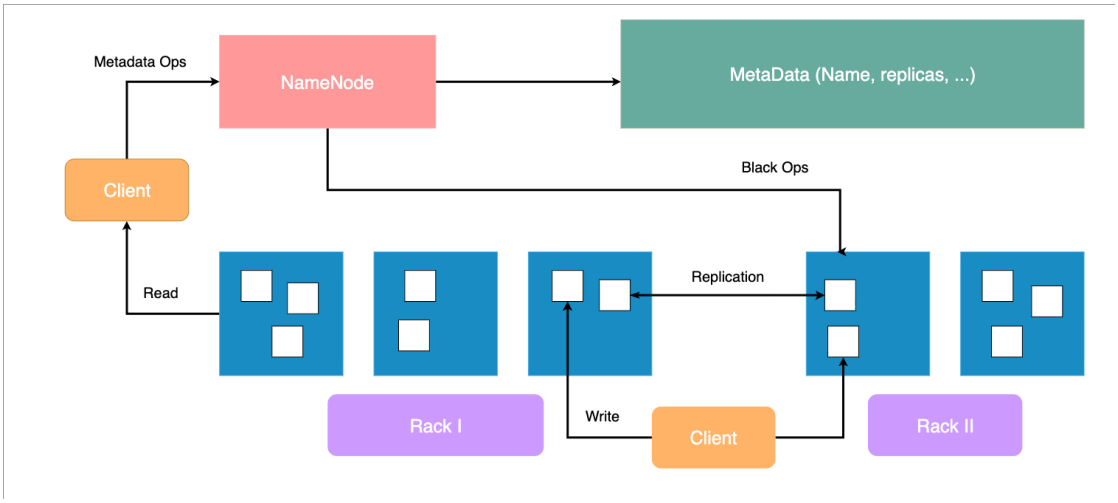


Figure 2.3: HDFS Architecture

Complementing HDFS was the MapReduce framework, which decomposed computational tasks into discrete mapping and reducing stages. In practice, this meant that large analytical problems could be partitioned into smaller subtasks, executed in parallel across the cluster, and then aggregated into results. This parallelism allowed enterprises to process volumes of data that were once unimaginable using traditional single-server or vertically scaled approaches. Together with YARN (Yet Another Resource Negotiator), which managed cluster resources and job scheduling, Hadoop created a complete ecosystem for distributed data storage and computation. The broader Hadoop ecosystem expanded to include tools that bridged the gap between low-level distributed computation and higher-level analytics. Hive provided an SQL-like query interface, translating user queries into MapReduce jobs, while Pig offered a procedural scripting environment for data transformation. Other components such as HBase introduced column-oriented storage for sparse, non-relational data, while orchestration tools like Oozie and coordination frameworks like ZooKeeper simplified management of complex data workflows. These extensions made Hadoop accessible to a wider range of practitioners, transforming it from a low-level computational engine into the foundation of early data lakes.

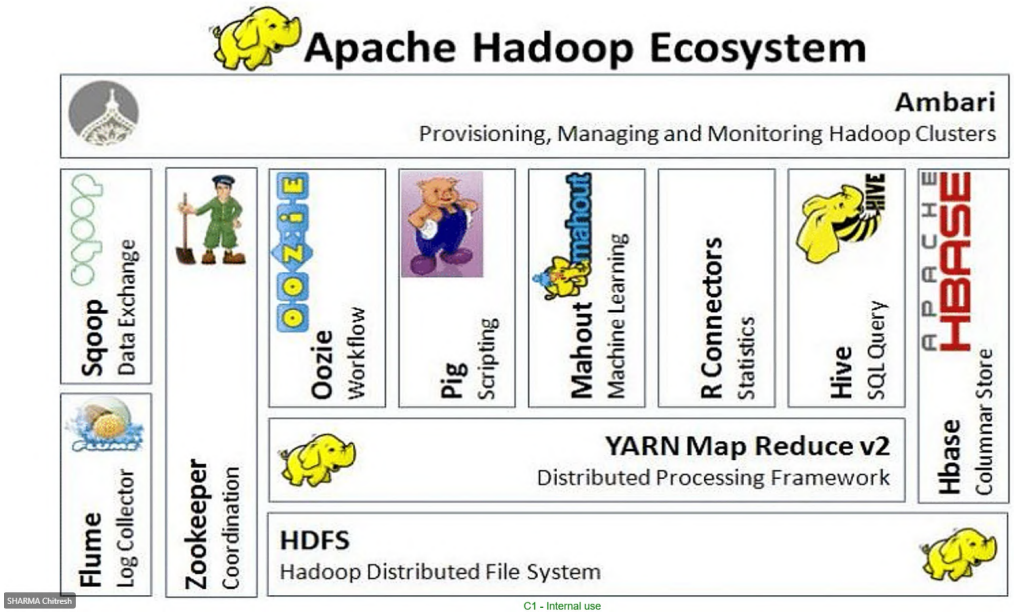


Figure 2.4: Hadoop Ecosystem

But Hadoop had limitations. Its batch-oriented nature made it ill-suited for real-time analytics, and the reliance on disk-based MapReduce introduced latency that hindered interactive analysis. Furthermore, its operational complexity and steep learning curve meant that only organizations with significant technical expertise could fully exploit its capabilities. These challenges opened the way for a new generation of distributed processing frameworks designed to address Hadoop's shortcomings.

Apache Spark emerged as a response to the limitations of MapReduce. Rather than persisting intermediate results to disk after each computational step, Spark introduced an in-memory processing model that dramatically reduced latency and improved performance for iterative workloads such as machine learning and graph processing. Spark's architecture was based on the abstraction of Resilient Distributed Datasets (RDDs), immutable collections of data that could be partitioned across clusters and operated on in parallel, with built-in fault tolerance through lineage tracking.

Unlike Hadoop MapReduce, Spark was not confined to a single processing paradigm. It was conceived as a unified analytics engine, supporting diverse workloads within a single framework. Spark SQL enabled structured query capabilities with an interface familiar to database practitioners; Spark Streaming extended the model to handle near real-time data ingestion and processing; MLlib provided a machine learning library capable of training models at scale; and GraphX facilitated graph analytics.

From an operational perspective, Spark introduced a simpler and more flexible programming model. Its APIs were available in multiple languages, including Scala, Java, Python, and R, which broadened its adoption among practitioners with diverse technical backgrounds. Furthermore, its interactive shell enabled exploratory analysis, allowing analysts to manipulate massive datasets without the overhead of writing and compiling MapReduce jobs. This interactivity, coupled with Spark's performance advantages, made it particularly attractive in environments where agility and rapid iteration were critical. Spark's architecture also lent itself naturally to cloud-native deployments. While Hadoop had originally been associated with on-premises clusters and complex hardware management, Spark could operate seamlessly in virtualized and managed environments, integrating easily with modern cloud storage systems and orchestration frameworks. Over time, Spark established itself as the de facto standard for distributed analytics, supported by a vibrant open-source community and adopted by leading technology companies as the processing engine behind their data platforms.
Although Spark is often presented as the successor to Hadoop, the relationship between the two technologies is more nuanced. Hadoop laid the conceptual and practical foundation for distributed computing, demonstrating that inexpensive clusters of commodity hardware could rival expensive proprietary systems in scale and reliability. Spark built upon these ideas but transcended Hadoop's constraints, enabling interactive, iterative, and real-time analytics that MapReduce could not achieve. The early Hadoop-based data lakes were pioneering in their ambition to collect and analyze raw, unstructured data at scale, but their complexity and limitations meant that many implementations failed to deliver sustained value. Spark provided the missing layer of usability, performance, and versatility, transforming the idea of the data lake from a theoretical construct into a practical architecture for modern analytics. Today, most cloud-native data lake implementations rely on Spark as the central processing engine, while Hadoop remains an important historical milestone that continues to influence distributed data system design.

2.4 Migration to the Cloud

Cloud migration is the movement of data, applications, and IT processes from on-premises infrastructure to cloud-based environments, represents a major shift in how organizations approach digital transformation. It is a strategic realignment of IT resources to achieve scalability, resilience, and cost efficiency.
The process typically begins with a careful assessment of existing systems, where organizations evaluate dependencies among applications, determine migration priorities, and establish a strategy aligned with business goals [7]. This preparatory

phase is essential, as it lays the groundwork for a seamless transition while minimizing risks such as downtime, data inconsistency, or compliance failures.
One of the primary motivations for moving away from traditional on-site infrastructure is the ability to respond to fluctuating demands with greater agility. The cloud offers a flexible, pay-as-you-go consumption model, which reduces reliance on large upfront capital expenditures for hardware. Instead of investing in infrastructure that may quickly become outdated, organizations shift their IT spending toward operational expenses, where resources can be scaled up or down as required. This model also relieves internal IT teams of maintenance and upgrade responsibilities, enabling them to focus more on innovation and strategic initiatives. Importantly, cloud adoption supports wider accessibility, allowing employees to collaborate effectively regardless of location and ensuring business continuity even during physical disruptions at the workplace.
Equally significant are the enhanced security and disaster recovery measures that cloud service providers typically offer. Encryption, identity and access management, continuous updates, and compliance certifications contribute to safeguarding sensitive data. This level of protection often surpasses what smaller or mid-sized organizations could implement with their own resources. Furthermore, disaster recovery capabilities built into cloud environments mean that critical applications and data can remain accessible during outages or system failures, thereby strengthening operational resilience.

The path to cloud migration, however, is not uniform and is heavily influenced by strategic choices regarding how applications are transferred. A "lift-and-shift" approach, also called rehosting, involves replicating existing systems in the cloud with minimal alterations. Its strength lies in speed and relatively low upfront cost, but it often fails to leverage the cloud's native capabilities, potentially leading to inefficiencies in the long term. A more balanced strategy, re-platforming, makes modest adjustments to applications to enhance compatibility and performance in the cloud environment without undertaking a full redesign. In contrast, refactoring — or re-architecting — represents the most transformative approach, where applications are fundamentally redesigned to fully exploit cloud-native features. This method demands substantial investment of time, expertise, and resources, but the long-term benefits include improved scalability, reliability, and readiness for future technological developments.
Decisions around deployment models are equally central to cloud adoption. Public clouds, provided by vendors such as AWS, Azure, or Google Cloud, allow multiple organizations to share infrastructure while benefiting from cost savings and rapid scalability. Nevertheless, concerns around security and compliance may limit their suitability for certain industries. Private clouds, either hosted on-premises or managed by third parties, offer greater control and customization but at higher expense and with added responsibility for infrastructure management. Hybrid

models have emerged as an attractive compromise, combining public and private cloud elements to balance flexibility, cost efficiency, and security. This approach enables organizations to allocate workloads strategically, running sensitive data in private environments while leveraging public cloud capacity for less critical operations. The migration journey itself unfolds in several phases.

Initial assessment and planning involve cataloging assets, mapping interdependencies, estimating costs, and identifying risks. Once the roadmap is defined, data migration and synchronization must be carefully managed to ensure integrity, minimize downtime, and maintain security throughout the transfer. Applications are then migrated according to the chosen strategy, configured to perform optimally in the new environment, and rigorously tested to validate functionality, performance, and compliance. Post-migration, optimization is crucial: organizations must adapt applications to benefit from cloud-native features such as autoscaling and load balancing, while instituting continuous monitoring to sustain performance and cost efficiency. Despite the advantages, cloud migration presents challenges that require strategic mitigation. Data security and regulatory compliance remain paramount, particularly when sensitive information is stored or transmitted across jurisdictions with varying legal requirements. Vendor lock-in also poses risks, as reliance on proprietary services can restrict flexibility and hinder future migration efforts. Furthermore, integrating cloud systems with legacy on-premises infrastructure often proves complex, necessitating robust middleware solutions and phased modernization. Performance optimization is another concern, given potential latency issues or mismatches between legacy application design and cloud environments. Finally, managing costs can be difficult due to variable usage patterns and intricate pricing models, underscoring the need for careful monitoring and resource optimization. The decision to migrate is often driven by specific organizational triggers. Businesses facing rapid growth, infrastructure obsolescence, rising capital costs, or operational inefficiencies may find cloud adoption particularly advantageous. Beyond selecting the appropriate migration strategy and deployment model, enterprises must cultivate in-house expertise, invest in training, and establish governance structures that ensure long-term sustainability.

Analysing the specific case of L'Oréal Group, the IT teams have undertaken the strategic decision to migrate all existing infrastructures and data to the cloud, including the platforms and tools that enable employees to monitor product sales and manage orders. This decision reflects a broader industry-wide trend, where the adoption of cloud technologies is increasingly viewed as a prerequisite for managing large-scale data systems and enabling advanced analytics. Cloud platforms have emerged as the natural enabler of this transformation. In the case of L'Oréal, Google Cloud Platform (GCP) was chosen as the technological backbone, offering a suite of services that combine computing, storage, data analytics, and machine learning within a single ecosystem. The migration to GCP was fundamental

redesign of how the company approached its information assets. The centralization was particularly significant for a multinational company whose operations span numerous markets, each generating its own stream of sales, supply chain, and customer-related data. Beyond centralization, the cloud offered scalability that on-premises solutions could not provide. L'Oréal operates in a highly dynamic industry, where fluctuations in demand, seasonality, and product launches produce variable loads on data systems. With GCP, the company can dynamically scale computational resources in response to these changing demands, avoiding the inefficiencies of over-provisioned hardware while maintaining the ability to process and analyse data at peak times. This elasticity ensures that business intelligence and operational systems remain responsive, supporting decision-making in near real-time. Another critical factor in the migration was the ability to integrate heterogeneous datasets. The capacity of GCP to integrate both structured and unstructured data created new opportunities for cross-functional analysis, enabling correlations that were previously obscured by the fragmentation of data sources. For example, sales performance could now be analysed in conjunction with online customer engagement metrics, providing a more comprehensive picture of market dynamics and consumer behaviour. Cost efficiency was also a decisive consideration. Traditional on-premises infrastructures required substantial upfront investments in servers, storage systems, and maintenance teams, while also generating ongoing expenses tied to hardware depreciation and software upgrades. By contrast, GCP's consumption-based model allowed L'Oréal to align infrastructure costs more closely with actual usage.

When discussing analytical databases, it is important to highlight platforms that have been specifically designed to meet the demands of large-scale data analysis in modern organizations. One such platform is Google BigQuery, which, although not always adopted in smaller projects, requires careful consideration given its capabilities.
BigQuery is a fully managed, AI-ready data warehouse built for high-performance analysis of massive datasets. Its serverless architecture eliminates the traditional burden of infrastructure management, allowing users to focus on solving analytical questions rather than configuring hardware and software resources. At its core, BigQuery is designed to process both structured and unstructured data efficiently. It supports open table formats such as Apache Iceberg, Delta, and Hudi, ensuring compatibility with diverse data pipelines. This flexibility extends to continuous ingestion through BigQuery streaming, which enables organizations to analyze real-time data as it flows into the system. The analytical engine is highly scalable and distributed, capable of querying terabytes of data in seconds and scaling to petabytes within minutes, a feat that underscores its value for large enterprises. BigQuery's architecture follows a principle of separation between storage and compute layers, a design that distinguishes it from legacy systems where these

functions often compete for the same resources. In traditional databases, read and write operations, query execution, and even administrative tasks such as managing permissions share the same resource pool, which can create bottlenecks and slow performance. BigQuery overcomes this limitation by decoupling storage and compute, each layer operating independently yet connected through Google's petabit-scale network. Governance and data management are also built directly into BigQuery. With tools like Dataplex Universal Catalog, users can curate data assets, trace data lineage, and assess data quality across their organization. We will discuss about these topics in the next chapter. From a usability perspective, BigQuery accommodates a wide spectrum of users. Business analysts can interact with it through the Google Cloud Console or via SQL, a language familiar to those experienced with relational systems. Developers and data scientists benefit from client libraries in Python, Java, Go, and other languages, along with REST and RPC APIs that support integration into broader workflows.

2.5 The Data Engineer as a Catalyst for Team Integration

The role of a data engineer does not exist in isolation; it sits at the intersection of technical expertise and organizational strategy. Within the corporate environment, data engineers act as the backbone, working in close collaboration with data scientists, analysts, business stakeholders, and software engineers. Their work ensures that raw, messy, and often heterogeneous data is transformed into structured, reliable, and accessible datasets that can be leveraged across the company.
One of the most important interactions is between data engineers and data scientists. While data scientists focus on building models, performing advanced analytics, and extracting insights, their productivity is heavily dependent on the availability of clean and well-structured datasets. In this sense, the work of a data engineer can be compared to that of an architect who designs and maintains the foundations of a building, while data scientists are the designers who create new features and applications on top of this foundation. A misalignment between these roles often leads to bottlenecks, highlighting the importance of communication and shared standards between teams.
Data engineers also collaborate with business intelligence analysts and product managers. Analysts require robust reporting systems and data pipelines that deliver reliable metrics, while product managers often need near real-time insights to guide strategic decisions. In many organizations, the data engineer assumes the responsibility of balancing these demands: building batch-processing pipelines for traditional reporting, while also architecting streaming solutions to accommodate real-time analytics. This dual responsibility illustrates how the role is both technical

and strategic, as decisions around pipeline architecture and storage formats can directly influence the company's agility and competitiveness.
The increasing complexity of the data ecosystem has also forced data engineers to interact with corporate leadership, particularly around issues of data governance, compliance, and cost optimization, with regulatory frameworks such as GDPR[1] in Europe, CCPA[2] in the state of California, United States, and PIPL[3] in China. The engineer is responsible for ensuring that systems adhere to privacy requirements and that data lineage can be traced. This elevates the data engineer to a role that involves risk management and compliance, making them a key figure in discussions around trust and transparency in data usage. As organizations mature, the role of the data engineer tends to evolve into a leadership or enabling function. Rather than handling all data tasks directly, engineers are often tasked with building platforms, services, and self-service tools that empower other professionals to access and manipulate data independently. This democratization of data access reduces dependency bottlenecks and promotes a data-driven culture, but it also requires engineers to adopt a mindset closer to that of a software product developer, focusing on usability, scalability, and user support. Engineers must be capable of understanding high-level strategic objectives, such as improving customer retention or optimizing supply chains, and then mapping those objectives to specific architectural choices—such as implementing a streaming data pipeline, restructuring a data warehouse schema, or adopting a new orchestration framework. In this sense, their role is both operational and consultative, bridging the gap between raw technical execution and business value creation. The evolution of data engineering as a discipline reflects this hybrid nature. It borrows heavily from software engineering in its emphasis on code quality, version control, automation, and scalability, but it also integrates aspects of business intelligence and database administration.

2.6 Machine Learning in the Cloud

The integration of machine learning with cloud computing represents an evolution in the way modern enterprises approach data-driven innovation. While machine

[1]GDPR is an EU law with mandatory rules for how organisations and companies must use personal data in an integrity friendly way.

[2]The California Consumer Privacy Act (CCPA) is a piece of data privacy legislation that applies to most businesses that process the personal data of California residents. The CCPA gives California residents a certain amount of control over the personal data that businesses collect about them.

[3]China's Personal Information Protection Law (PIPL), a comprehensive privacy law intended to protect the personal information of persons in Mainland China, the continental landmass under the direct control of the People's Republic of China (PRC).

learning has long promised the ability to extract meaningful patterns from vast and heterogeneous datasets, its real-world application has historically been constrained by the significant computational resources required for both training and inference. On-premises infrastructures often proved insufficient to accommodate the combination of massive storage capacity, distributed computation, and GPU acceleration that advanced models—particularly deep learning architectures—demand.
Cloud computing fundamentally removes these barriers by offering virtually unlimited processing power and storage, that can be provisioned elastically according to project requirements. This allows organizations both to handle the scale of big data and to iterate quickly, experimenting with multiple models, architectures, and parameters without being limited by local infrastructure. The result is a shorter time-to-value, where insights and predictive capabilities can be deployed into production far more rapidly than in a traditional IT environment.

In the specific case of L'Oréal, the benefits of this paradigm are both technical and strategic. The beauty industry operates in an environment characterized by volatile consumer preferences, globalized supply chains, and the constant launch of new products across diverse markets. Machine learning models hosted in the cloud enable the company to respond to these dynamics with unprecedented agility. For example, demand forecasting models trained on sales and seasonal data can be continuously updated as new information flows into the system, ensuring that inventory management decisions are timely and accurate. Similarly, customer personalization initiatives — ranging from product recommendations in e-commerce platforms to personalized marketing campaigns — can leverage scalable ML services that analyse millions of data points across regions and customer profiles. By relying on the cloud, L'Oréal avoids the need for heavy upfront investments in specialized infrastructure, instead benefiting from a flexible "pay-as-you-go" model that matches expenditure with real usage while maintaining access to cutting-edge ML capabilities.
Another significant advantage of hosting machine learning in the cloud lies in its democratization of access. Whereas advanced AI techniques were once the exclusive domain of highly specialized teams, cloud providers now offer pre-built APIs, drag-and-drop interfaces, and AutoML services that allow a much broader set of employees—from business analysts to marketing managers—to embed machine learning into their workflows. For a global enterprise such as L'Oréal, this democratization translates into an organization-wide capacity to transform raw data into actionable insights without requiring every business unit to maintain dedicated data science teams. The synergy between centralized data engineering pipelines and decentralized business applications ensures that insights are technically robust and aligned with operational needs across departments.
By coupling scalable computation with advanced predictive modeling, L'Oréal can strengthen its ability to anticipate consumer behaviour, optimize supply chains, and tailor its marketing strategies with a level of precision and speed that cannot

be achieved through traditional infrastructures.
Machine learning, and generally artificial intelligence, has opened up a wide range of opportunities for enterprises seeking to optimize their operations, and create more personalized services, and its applications are now visible across industries as diverse as finance, retail, logistics, and healthcare.
In the financial sector, machine learning has become indispensable for fraud detection, as models trained on large volumes of historical and real-time transaction data are capable of identifying anomalous activities that may indicate fraudulent behavior. Unlike rule-based systems, these models continuously learn and adapt as fraud tactics evolve, thereby reducing both financial losses and the frequency of false positives that inconvenience legitimate customers. Banks such as HDFC and ICICI have already implemented supervised anomaly detection and time-series forecasting techniques to detect suspicious transactions within milliseconds and to improve the accuracy of credit risk assessments. Beyond fraud detection, machine learning also plays a critical role in enhancing financial forecasting, budgeting, and risk modeling, with institutions like Bajaj Finance using these tools to better predict portfolio risks and improve credit offerings. Similarly, stock market trading is being reshaped by the adoption of reinforcement learning and algorithmic trading models that can process market data, news, and economic indicators in real-time to inform trading strategies and execute high-frequency trades far beyond the capacity of human traders.
In business operations, machine learning has revolutionized supply chain management and demand forecasting, allowing enterprises to predict customer demand with greater precision, optimize inventory levels, and ensure that products are available at the right time and place. Retailers have successfully applied these techniques to anticipate fluctuations in demand, particularly during peak seasons, thereby reducing both excess inventory and missed sales opportunities. At the heart of these efforts are models such as random forests and Prophet, which analyze past sales data, seasonal variations, and external market trends to generate accurate forecasts. Machine learning further extends into inventory management, where predictive analytics allows businesses to reduce costs associated with overstocking or stockouts.
For instance, Amazon's global supply chain relies heavily on machine learning to dynamically adjust stock levels across its warehouses, ensuring timely fulfillment of customer orders while optimizing logistics efficiency [8]. Predictive analytics also strengthens decision-making across multiple industries, as companies are increasingly relying on models to identify customer behavior patterns, sales trends, and potential risks, with companies like Netflix leveraging recommendation algorithms to enhance user engagement and Flipkart using machine learning to optimize marketing campaigns and predict shopping behaviors. Logistics providers, including Uber and DHL, demonstrate how route optimization powered by clustering and

heuristic optimization algorithms can cut travel time, reduce fuel consumption, and lower operational costs, resulting in improved customer satisfaction and sustainability.
Healthcare represents another area where enterprises can leverage artificial intelligence to achieve transformative results, particularly in diagnostics and patient care. Machine learning algorithms such as convolutional neural networks (CNNs)[4] have proven highly effective in analyzing complex medical data, including radiographs and MRI scans, to detect diseases at early stages and support doctors in making faster and more accurate diagnoses. Institutions like AIIMS Delhi are already applying ML-powered systems to identify early signs of lung cancer in X-rays, while Medtronic integrates predictive models to detect cardiovascular issues in real-time during checkups. Beyond diagnostics, speech and image recognition systems are being deployed to transcribe doctor-patient conversations in real-time or to automate the analysis of medical imaging, thereby reducing the burden on healthcare professionals. Facial recognition, another application of machine learning, is also being introduced in hospitals for patient identification, improved security, and even monitoring of patient health status through analysis of facial expressions.

Lastly, Generative AI has emerged as one of the most transformative technologies in the corporate landscape. Unlike traditional AI systems that focus on classification, prediction, or optimization, generative AI is capable of producing original content—whether in the form of text, images, code, or even structured data—based on patterns learned from vast datasets. In a corporate context, this capability can be harnessed across multiple domains. Marketing teams use generative models to craft personalized campaigns at scale, dynamically tailoring messages to customer segments while maintaining brand voice. In product development, generative AI accelerates prototyping by suggesting design alternatives or generating synthetic data to test new features. Within operations, it streamlines internal processes by automatically drafting reports, generating documentation, or assisting employees through conversational agents trained on proprietary knowledge bases. Moreover, generative AI supports strategic decision-making by simulating scenarios, producing insights from unstructured datasets, and integrating seamlessly with analytical platforms such as BigQuery to enrich business intelligence with predictive and generative layers. Its adaptability and capacity to augment human creativity make it a catalyst for innovation, fundamentally reshaping how enterprises long-term growth.

[4]A convolutional neural network is a type of feedforward neural network that learns features via a kernel optimization. Convolution-based networks are the de-facto standard in deep learning-based approaches to computer vision and image processing, and have only recently been replaced—in some cases—by newer deep learning architectures such as the transformer.

2.7 Technologies

Apps are becoming increasingly structured, and some features and technologies have to be adopted to reduce complexity, increase efficiency and optimise workflows. Continuous integration (CI) consists of frequent, automatic integration of code changes into a shared source code repository[9]. Continuous delivery and/or continuous deployment (CD) is a two-part process during which code changes are integrated, tested, and deployed. Automated delivery does not fully correspond to continuous deployment, which automatically releases updates to the production environment. The CI/CD method helps organisations reduce the risk of bugs or code errors, while enabling continuity in the software development and update cycle. It generally automates the steps (usually performed manually by human operators) required to extract new code from an application and use it in production. This minimises downtime and speeds up code releases. In addition, by accelerating the integration of updates and code changes, user suggestions can be incorporated more frequently and efficiently. By separating development and production environments, CI/CD ensures that new features can be tested in controlled conditions before reaching end users, thereby reducing risks and preserving system stability.

The systematic approach to automation is complemented by infrastructure-as-code solutions such as Terraform, an open-source tool developed by HashiCorp, that enables Infrastructure as Code (IaC) and manages cloud infrastructure.
Terraform's importance lies in its contribution to the industrialization of infrastructure, replacing manual configurations with version-controlled code that can be shared, audited, and scaled across projects. This brings immediate advantages. First, it eliminates human error: instead of manually creating a dataset and risking inconsistent naming or permissions, the configuration enforces organizational standards automatically. Second, it ensures reproducibility: the same Terraform module can be applied across environments (development, staging, production), guaranteeing that resources are provisioned in a predictable and uniform way. The motivation for industrialization stems from the challenges of scale. In a large organization such as L'Oréal, cloud projects are not static; they evolve as new teams, datasets, and analytical needs emerge. Without automation, maintaining consistency across dozens of projects and regions would be unsustainable. Industrialization allows the organization to move beyond ad-hoc experimentation—where resources are created on the fly—to a sustainable governance model, where policies for cost control, security, and monitoring are built directly into the provisioning process.
Another key advantage is collaboration and transparency. Because Terraform code is versioned on GitHub, every infrastructure change is visible, reviewed, and documented. Moreover, integrating Terraform into CI/CD pipelines means that deployments can be automated, tested, and enforced with organizational policies,

further strengthening the overall governance framework.

Alongside infrastructure and deployment, data integration remains a crucial concern, where the distinction between ETL and ELT approaches shapes how organizations handle growing volumes of information. ETL stands for "Extract, Transform, Load" and it emphasizes data transformation prior to storage while ELT, "Extract, Load, Transform", leverages the computational power of modern warehouses, loading raw data first and transforming it afterward to maximize flexibility. Tools such as Fivetran have become prominent in this context, a technology to fully manage data integration, automating the part of extracting and loading. Fivetran provides pre-built connectors to a wide variety of sources, such as databases, SaaS applications, and cloud services, and enables their seamless integration into a centralized data warehouse or data lake. From a technical perspective, once a connector is configured, the system automatically manages schema changes, incremental updates, and error handling without requiring significant manual intervention.

Once data is ingested and transformed, its value depends largely on how it is communicated. Studies in cognitive science have consistently demonstrated that humans respond more effectively to visual stimuli than to any other type of sensory input. The human brain is estimated to process visual information approximately 60,000 times faster than textual information, and it is further noted that around 90% of the information transmitted to the brain is visual in nature[10]. This cognitive predisposition underscores the importance of data visualisation in contemporary contexts, particularly in business environments where vast quantities of information are generated both internally and externally. Data visualisation serves as a critical mechanism for interpreting this data in meaningful ways. Data visualisation can be defined as the graphical representation of data with the purpose of facilitating communication between abstract information and the user. This process relies on mapping techniques that allow quantitative and qualitative information to be transformed into graphical formats. Through the use of statistical graphs, diagrams, infographics, and related tools, numerical data can be encoded visually in the form of dots, lines, or bars to convey quantitative messages in a more intuitive manner. Patterns, correlations, and anomalies that might remain hidden in traditional tabular reports or spreadsheets become more readily identifiable when visualisation techniques are employed.
In recent years, data visualisation tools have evolved considerably beyond the basic charts and tables historically associated with spreadsheet software such as Microsoft Excel. Modern approaches encompass a wide variety of graphical forms including infographics, scales, geographical maps, sparklines, heat maps, histograms, and pie charts. Increasingly, these visualisations are interactive, and it has become particularly significant in the context of Big Data and the Internet of Things (IoT), where the exponential growth of data generated through digital interactions

and sensor technologies has created new analytical and ethical challenges. The significance of data visualisation is especially pronounced in the domain of Business Intelligence (BI), where it has emerged as the de facto standard for presenting and interpreting information. Visualisation tools have played a transformative role in democratising access to data and analytical insights. Unlike traditional statistical analysis software or earlier iterations of BI systems, contemporary data visualisation platforms are typically more user-friendly, which has encouraged widespread adoption across multiple lines of business without requiring specialised IT support.

Beyond visualization, the rise of machine learning in enterprises has given way to MLOps, a discipline that extends DevOps principles to the lifecycle of machine learning models. To better understand its role it would be instructive to compare the creation of machine learning models with conventional application development. In traditional software engineering, the process typically begins with the review of a specification, followed by the design of data structures that should remain stable once the coding phase begins. Implementation then proceeds with coding as the primary activity, accompanied by the creation of unit and end-to-end tests to verify correctness.
By contrast, the development of machine learning models follows a markedly different trajectory. The process begins with an understanding of raw data and the predictive task at hand. Although coding is required to implement neural networks or configure algorithms, the predominant activity is experimentation. Throughout repeated experiments, the design of the data, the structure of the model, and the hyperparameters undergo frequent modifications. Each experiment generates metrics to evaluate model performance during training, as well as against validation and test datasets. These metrics serve as indicators of model quality. Once a model achieves satisfactory performance, it must be packaged and deployed in order to be integrated into an application.
MLOps has emerged as a response to the distinctive requirements of this workflow. It encompasses a set of practices and tools designed to streamline experimentation, collaboration, and deployment. Modern tools can supply runtime environments for experiments, manage model packaging, and facilitate deployment. They may also include features such as integration with modern data lakes for handling structured and unstructured experimental data, mechanisms for collaboration across teams, model registries for maintaining version histories, erverless functions for containerised experimentation, and pipeline orchestration for both data and training processes.
However, the successful implementation of MLOps is closely linked to the underlying infrastructure, particularly in environments that rely on Graphics Processing Units (GPUs) for model training. A recurring challenge in such contexts is the so-called “Starving GPU Problem.” This occurs when storage solutions or network

throughput are insufficient to provide training data at a rate that can match GPU computational capacity. The consequences of this bottleneck are easily observable: GPUs fail to achieve high utilisation, and overall training time becomes dominated by input/output (I/O) delays. The challenge is exacerbated by the rapid advancement of GPU technology[11].
The implications of these hardware advances are significant. GPU performance, measured in teraflops, highlights the scale at which data must be delivered to fully exploit available compute power. Increases in GPU memory capacity enable larger batch sizes during training, leading to greater throughput demands. Likewise, proportional increases in memory bandwidth ensure that data transfer between GPU memory and compute cores does not constitute a limiting factor. Together, these enhancements increase the volume and velocity of data that must be supplied, magnifying the risk of encountering the Starving GPU Problem if storage and network systems do not keep pace.

Chapter 3

Methodology

Chapter Three marks a key stage in my research journey, as it transitions from the theoretical exploration of artificial intelligence and cloud computing concepts into their practical application through three distinct projects. Each of these projects is representative of a different set of challenges, objectives, and technological environments, allowing me to engage with both experimental prototypes and solutions that can be directly translated into enterprise contexts.

The mission that drives this chapter can be summarized in the design and implementation of these projects, where I will highlight the technical choices and the motivations that shaped them, the methodologies applied, the collaborative dynamics with teams, and the results obtained. This approach offers a comprehensive view of how complex infrastructures and machine learning models can be integrated into real workflows, and how cloud-based platforms can support scalability, automation, and innovation.

As already mentioned before, Google Cloud Platform provides a robust and flexible environment for building, deploying, and managing applications at scale, and understanding its core components is essential before delving into more advanced services. At the foundation of GCP's organizational structure lies the concept of projects, which serve as isolated workspaces where resources are created, managed, and billed. Each project has its own unique identifier and encapsulates all associated services, from virtual machines and storage buckets to databases and machine learning models. This separation not only simplifies cost tracking and resource allocation but also ensures that permissions, configurations, and policies remain consistent within each environment.

A crucial element in maintaining security and access control within projects is Identity and Access Management (IAM). IAM allows administrators to define who has access to which resources and what level of permission they are granted, using predefined or custom roles. By implementing IAM policies, enterprises can adopt the principle of least privilege, thereby reducing risks while maintaining

flexibility across teams. This is particularly important in collaborative settings, where developers, data scientists, and system administrators may require different levels of access to the same infrastructure.

Alongside projects and IAM, GCP offers several other essential tools for getting started. The Cloud Console, with its graphical interface, and the Cloud SDK, with its command-line tools, provide developers with different ways to interact with resources, enabling both manual provisioning and scripted automation.
For networking, services like the Virtual Private Cloud (VPC) establish secure and customizable networks where resources can communicate, while Cloud Monitoring and Cloud Logging offer observability, helping teams track performance metrics and troubleshoot issues effectively. Storage needs are addressed through a spectrum of services, ranging from Cloud Storage for unstructured data to managed databases such as BigQuery for large-scale analytical workloads. Together, these foundational elements create a cohesive ecosystem that allows users to start small with individual experiments and scale up to enterprise-grade systems.

3.1 A Dashboard for Data Visualization

3.1.1 Motivations

The first project is about the creation of an interactive dashboard through Looker Studio. This initiative emerged from the need to transform raw data into meaningful visual insights, to monitor performance and detect trends with clarity and immediacy.

The main GCP project is the Data Exploration Environment, conceived to provide a dedicated production-ready space where Data & Analytics teams could engage in exploration, experimentation, and the validation of strategic assumptions. Within this framework, the initiative was not merely an exercise in resource deployment but a structured response to the growing need for agility, efficiency, and cost awareness. The objective of the environment is also to establish a consistent and scalable process that empowers data professionals to design proofs-of-concept, evaluate their feasibility, and ultimately translate these findings into actionable insights that can guide organizational strategy.

My involvement in this initiative centered on the development of a dashboard, undertaken under the guidance of the Data Tech Lead, with the explicit aim of enhancing visibility into the expenses and resource usage associated with BigQuery. This was a critical task because, although L'Oréal leverages Google Cloud as a unified platform for data storage and analytics, the pay-as-you-go pricing model demands careful monitoring to prevent unnecessary overhead. Without such a mechanism, teams risk incurring unforeseen costs, especially when working with large volumes of data that require frequent querying and optimisation of storage.

The creation of the dashboard directly addressed this issue by introducing a tool capable of aggregating and presenting financial, operational, and resource metrics in a single, user-friendly interface. The resulting artifact was structured around three distinct perspectives—costs, usage, and storage—allowing users to simultaneously assess financial performance, operational efficiency, and resource consumption. This multidimensional approach was designed to deliver a comprehensive understanding of both project-level and service-level dynamics. Beyond tracking current expenses, the dashboard also enables stakeholders to identify inefficiencies, anticipate budget deviations, and align infrastructure usage with business priorities. To ensure a systematic rollout, each of the three sections was developed through a phased approach consisting of data extraction, dashboard design, and industrialization. In this chapter, we will take an in-depth look at how data extraction and industrialisation have been implemented, while dashboard design will be discussed in the next chapter in the context of results analysis.

3.1.2 Project Specifications

Before proceeding with the implementation of data extraction and industrialisation, it is essential to define the functional and technical specifications that guided the development of the dashboard. The Data Exploration environment was conceived as a "sandbox": a controlled ecosystem where teams from various domains—principally Data Engineering, but later Retail and E-commerce—could conduct experiments and launch proofs-of-concept. While this flexibility is crucial for innovation, it poses a significant risk of resource sprawl. Therefore, the primary objective of the dashboard is to monitor this usage strictly, ensuring that the operational freedom granted to internal L'Oréal users does not translate into uncontrolled expenditure. To achieve this, the system was designed around specific data sources, analytical requirements, and technical constraints.

Unlike traditional financial reporting tools that rely directly on the Cloud Billing Export, this dashboard had to use a proxy approach to estimate costs based on resource consumption. Two primary data sources were identified to cover the different aspects of the platform. The first is the BigQuery Information Schema, which is used to retrieve metadata regarding the static footprint of data, effectively providing snapshots of table sizes and row counts. The second is Cloud Logging, which captures dynamic activity and usage. Since the actual Billing Export was not connected for this specific scope, costs are estimated by analyzing the `total_bytes_billed` recorded in the logs of every executed job.

Regarding granularity, the analysis is conducted at two distinct levels. At a high level, metrics are aggregated by Project to evaluate the overall health and consumption of specific exploration zones (e.g., `emea-dataex-fr`). However, to pinpoint specific inefficiencies, the data is also analyzed at the Job level. In

BigQuery, a "Job" represents a single atomic unit of work, such as a SQL query, a data load, or an export operation. Granularity at this level is strictly necessary to identify whether a budget deviation is caused by general usage or by a single, poorly optimized query consuming a disproportionate amount of resources.

In terms of operational efficiency and usage analysis, the focus shifts to the intensity of the platform's utilization. The metrics are strictly defined by the volume of data processed (measuring `total_bytes_billed`, which is the primary driver for query costs) and the query volume. The visualizations must highlight spikes in daily usage, allowing administrators to correlate high consumption periods with specific events or user activities.

Finally, the storage analysis addresses the persistent cost of data retention. The requirements for this section involve measuring the volume of space occupied by tables and materialized views, expressed in Gigabytes. Beyond mere volume, the dashboard must monitor the number of active tables and views per project to detect object proliferation. Crucially, it must also visualize the growth trend of storage over a defined window, such as a three-month rolling period, to identify the steady accumulation of stale data.

3.1.3 Functional Requirements

From a financial monitoring perspective, since the environment is budget-constrained, the dashboard must provide immediate visibility into financial deviations. The requirements stipulate the ability to identify "Top Spenders"—ranking users and projects based on their estimated daily consumption—to enable targeted optimization actions. Furthermore, the system must provide a direct comparison between the estimated cost and the assigned monthly budget, alongside a Month-to-Date (MTD) trend view. This cumulative view of expenses throughout the month is essential for predicting end-of-month projections and detecting anomalies early.

To ensure usability and relevance for different stakeholders, the dashboard must adhere to strict technical constraints regarding interactivity and data freshness. The interface is designed to allow dynamic slicing of data through mandatory filters. These include a Project ID selector, a User Email filter to audit specific team members, and a flexible Time Range selector. Users must be able to switch seamlessly between preset views—such as the Current Month, Last Week, or Today—and custom date ranges.

Given the need for prompt reaction to consumption spikes, the data pipeline is required to refresh at least four times per day, effectively every six hours. Furthermore, to ensure readability across both technical and non-technical audiences, all volume metrics must be standardized, converted, and presented in Gigabytes (GB) or Terabytes (TB) where appropriate. Defining these specifications was a prerequisite for the subsequent phases of the project: the extraction of raw data

and the industrialisation of the infrastructure.

3.1.4 Data Extraction

The first stage of the project focused on extracting the necessary information in a structured, tabular format, since a dashboard without reliable and well-prepared data would be of little use. To achieve this, we needed to capture two sources of information: storage consumption and query usage. For the storage component, most of the information from all domains is stored in BigQuery in tabular format, so the challenge was to determine how much space each table occupied, as this directly impacts both costs and resource allocation.

The first solution was to query INFORMATION_SCHEMA views, which provide system-defined metadata about datasets, tables, and other resources. By leveraging views such as TABLES and TABLE_STORAGE, we were able to retrieve essential attributes including the table type (table, view, or materialized view), the number of rows, and, most importantly, the size of each table. To ensure that the information remained up to date, a scheduled query was set to run daily, functioning as a lightweight yet systematic process that continuously extracted and stored metadata in a dedicated table of destination. This daily ingestion enabled the monitoring of project growth, identification of oversized tables, and ultimately, the detection of inefficiencies in data storage.

However, this approach quickly revealed itself to be problematic due to its excessive computational cost. The difficulty lay not in accessing the data itself but in the way BigQuery processed the query behind the scenes. Although the INFORMATION_SCHEMA views are designed to expose metadata, BigQuery still performs complex calculations when resolving storage statistics, and this results in significant data scans. In practice, what was expected to be a lightweight metadata query ended up generating substantial analysis costs. A closer inspection demonstrated just how unsustainable this approach was. To put this into perspective, let's consider an approximation: a single query accumulated around 670 gigabytes. Factoring in an average of 30 days per month and a repetition of 10 queries for each external project analysis, the computation cost escalates significantly. The cost calculation unfolds as follows: 10 projects * 670GB * 30 days, dividing by 1024 to obtain the result in terabytes, it gives 196,3TB per month. Knowing that BigQuery currently charges 5 dollars (USD) per terabyte processed in its Enterprise version, the price rises to 981.45 dollars (USD) per month, a figure that is disproportionately high given the nature of the task. This is a remarkable expense for what should have been a simple and recurring monitoring task, and it directly contradicted the objective of the project, which was precisely to reduce unnecessary costs and provide transparency into cloud expenditures.

The fundamental issue here was not the lack of functionality—since INFORMATION_SCHEMA.TABLE_STORAGE indeed contained the required attributes but rather the inefficiency of the method. Querying metadata in this manner forced BigQuery to perform exhaustive calculations on underlying resources, making the solution impractical at scale. Consequently, it became clear that an alternative method had to be devised: one that could produce equivalent insights without incurring such high costs. The search for a workaround thus became essential, with the goal of striking a balance between precision in monitoring and efficiency in query execution.

Since the query was intended to run on a daily schedule, this meant carefully analyzing the limitations of INFORMATION_SCHEMA and systematically applying cost optimization best practices to strike a balance between completeness and efficiency. One limitation I had to account for was that INFORMATION_SCHEMA queries are only supported in GoogleSQL syntax, not legacy SQL. This was not a major obstacle in itself but required consistency across all scripts and careful attention to query structure. The optimization process therefore centered around exploiting BigQuery's rules for how queries are billed. The first strategy that could be involved reducing the volume of data scanned is through partitioning. Partitioned tables are particularly advantageous when dealing with time-based data because they restrict processing to only the partitions relevant for a given query. This meant avoiding full-table scans and significantly lowering the number of bytes processed.
Alongside partitioning, we could also employ clustering, which organizes data within each partition by sorting it according to selected columns, making it easier for BigQuery to skip unnecessary rows and further minimizing scan costs. Another major source of inefficiency is the careless use of SELECT *. Although convenient during testing, this practice leads to querying all columns—even those irrelevant to the analysis—causing needless processing and higher bills. By explicitly selecting only the required metadata fields (such as table name, row count, and storage size), the final query was able to process only the data that truly mattered.
Finally, I leveraged BigQuery's scheduled queries functionality to ensure the process was automated but still cost-conscious. Scheduled queries are inherently more efficient than rerunning ad hoc queries, as they can take advantage of caching mechanisms when the data remains unchanged. This guaranteed that the query could be executed daily, producing updated results without unnecessary duplication of effort or additional cost. By carefully combining these optimization strategies, the final query managed to deliver the required metadata in a lean, efficient, and financially sustainable way. The following query is the result of attempts to find the most efficient solution.

```
1 FOR record IN (
```

```
2     SELECT * FROM UNNEST(SPLIT('emea-dataex-fr-emea-pd,emea-dataex
    -bnl-emea-pd,emea-dataex-ibe-emea-pd,
3     emea-dataexzone-gbl-emea-pd,emea-dataex-pol-emea-np,emea-
    dataex-ukr-emea-np,emea-dataex-ukl-emea-np,emea-dataex-dach-
    emea-pd,
4     emea-dataex-nord-emea-pd,emea-dataex-grc-emea-np,emea-dataex-
    cze-emea-pd', ',')) AS project)
5 DO
6     EXECUTE IMMEDIATE CONCAT('CREATE OR REPLACE TEMP TABLE
    proj_datasets AS SELECT schema_name FROM ', record.project,
7     '.region-eu`.INFORMATION_SCHEMA.SCHEMATA');
8
9     IF EXISTS(
10        SELECT schema_name FROM proj_datasets LIMIT 1
11    ) THEN
12        EXECUTE IMMEDIATE CONCAT(
13            'INSERT INTO emea-monitoring-gbl-emea-pd.
    d_storage_dataex_eu_pd.t_storage_dataex (',
14            (SELECT
15                STRING_AGG(
16                    CONCAT(
17                        'SELECT *,',
18                        '"project_id AS table_catalog,',
19                        '"dataset_id AS table_schema,',
20                        '"table_id AS table_name,',
21                        'IF(type = 1, "BASE TABLE", "VIEW") AS
    table_type,',
22                        'TIMESTAMP_MILLIS(creation_time) AS
    creation_time,',
23                        'TIMESTAMP_MILLIS(last_modified_time) AS
    last_modified_time,',
24                        'CURRENT_TIMESTAMP() AS last_update,',
25                        '"row_count,',
26                        '"size_bytes"',
27                        'FROM ', record.project, '.', schema_name,
     '.__TABLES__'
28                    ),
29                    'UNION ALL \n'
30                )
31            FROM proj_datasets
32            ),
33        ')';
34    );
35 END IF;
36 END FOR;
```

Generally, it systematically scans predefined projects, extracts table metadata,

and consolidates this information into a centralized monitoring table. The first part of the scheduled query creates a dynamic list of L'Oréal projects to be scanned. The project names convention have the structure `emea-dataex-<country>-emea-<env>` where country codes can iterate over:

`fr` for France; `bnl` Benelux (Netherlands, Belgium, Luxembourg); `ibe` for Iberia (Spain and Portugal); `pol` for Poland; `gbl` for Global; `ukl` and `ukr` for United Kingdom of Great Britain and Northern Ireland; `dach` for DACH Area (Germany, Austria and Switzerland); `nord` for Nordic Countries (Norway, Finland, Iceland, Denmark and Sweden); `grc` for Greece and Cyprus; `cze` for Czech Republic.

These are the beta countries for which the Data Exploration (dataex) project was launched in emea geographic region (Europe, Middle East, Africa). And env designates the environment, in this case production rather than non-production or development. Through the SPLIT and UNNEST, the script converts a comma-separated string into an array of project names and loops with DO instruction. The purpose is to dynamically discover all datasets within each project, this is done through the statement EXECUTE IMMEDIATE: you can use it to issue SQL statements that cannot be represented directly in PL/SQL, or to build up statements where you do not know all the table names, WHERE clauses, and so on in advance.

Then the script creates a temporary table, proj_datasets, containing the list of datasets within a given project. The INFORMATION_SCHEMA.SCHEMATA view is used as the authoritative metadata source for datasets, while the explicit regional qualifier (region-eu) ensures that only resources located in the European multi-regional environment are retrieved. Once the datasets have been identified, a conditional check is performed to validate whether any exist in the current project before proceeding further. This control mechanism plays a dual role: it enhances efficiency by avoiding unnecessary operations in projects with no datasets, and it provides robustness by preventing execution errors caused by the absence of resources.

Following the discovery phase, the process transitions to metadata collection and aggregation. A dynamic query string is constructed using the STRING_AGG function, which concatenates multiple SELECT statements into a single query, joined by UNION ALL. This query targets BigQuery's internal ___TABLES___ metadata table within each dataset. Metadata fields collected include:

- `project_id`: Source project identifier;
- `dataset_id`: Dataset (schema) name within the project;
- `table_id`: Individual table name;
- `table_type`: Distinguishes between base tables (type = 1) and materialized views (type != 1);

- `creation_time`: When the table was originally created (converted from milliseconds);
- `last_modified_time`: Last modification timestamp (converted from milliseconds);
- `last_update`: is computed at runtime to capture the execution timestamp of the monitoring script;
- `row_count`: Number of rows in the table;
- `size_bytes`: Storage size occupied by the table;

The consolidated metadata is then inserted into a centralized monitoring repository: `emea-monitoring-gbl-emea-pd.d_storage_dataex_eu_pd.t_storage_dataex`. This repository resides in a dedicated global project designed for monitoring purposes, within which the European dataset serves as the designated collection point for all storage-related metrics. By centralizing the results of the discovery and metadata collection process, the solution enables systematic tracking of assets across projects and regions. From a technical perspective, the process relies on UNION ALL operations to combine results from multiple datasets into a unified result set.
The business value of this architecture is multifold. From a data governance standpoint, the system enables automatic cataloging of assets across the organization, facilitating compliance monitoring and supporting access control reviews. From a cost management perspective, it provides visibility into storage utilization, helping to identify oversized or unused tables, forecast growth, and allocate budgets more accurately. Architecturally, the solution demonstrates scalability by handling multiple projects concurrently and adapting to variations in dataset size and type.
Error handling is embedded through conditional processing and dynamic SQL generation, allowing the system to accommodate projects with different structures or missing resources without interruption.

While the storage-related metrics could be collected directly from BigQuery's INFORMATION_SCHEMA, usage statistics required a different approach.
For this purpose, a complementary project—internally known as Monitoring—was employed. This project was specifically designed to capture and centralize logs of user activity across the EMEA data platform, including queries executed against BigQuery. The cornerstone of this implementation is the Google Cloud Logging infrastructure, which records detailed event logs whenever users or services interact with BigQuery. To make these logs queryable, the Monitoring project provisions a folder-level logging sink using the google_logging_folder_sink Terraform resource. This sink acts as a streaming pipeline: it intercepts relevant log entries at the folder level (encompassing all projects contained within that folder) and exports them

into a dedicated BigQuery dataset for long-term analysis. Once configured, the logging sink continuously streams usage logs as they are generated. This ensures that the dataset in BigQuery reflects user activity in near real time. Each log entry corresponds to a job executed on BigQuery, with granular details such as the job type (query, load, extract), the identity of the user or service account executing it, the timestamp of execution, and—most critically—the number of bytes billed by the job. This last metric is particularly important for cost attribution and optimization, since BigQuery pricing is directly tied to the volume of data processed.
From a practical standpoint, the exported logs are stored in a structured table within the Monitoring project's BigQuery dataset. Each row of this table corresponds to an individual query job, containing fields such as: `job_id`, a unique identifier for the query execution; `user_email`, the account responsible for running the job; `query_text`,the SQL statement executed (subject to access restrictions); `start_time` and `end_time`, provide precise execution timestamps, useful for performance monitoring; `total_bytes_billed`, the volume of data processed, which is directly tied to query costs; `project_id` and `dataset_id`, are the context of the data queried.

Here is an example of how a BigQuery usage log entry exported into the Monitoring dataset might look. Logs are structured in JSON-like form.

```
{
    "job_id": "bqjob_r1d8f12c_1234567890",
    "user_email": "amarino@loreal.com",
    "query_text": "SELECT customer_id, SUM(purchase_amount) AS
   total_spent FROM `retail_sales.transactions` WHERE
   purchase_date >= '2023-01-01' GROUP BY customer_id",
    "start_time": "2024-06-12T09:14:23.456Z",
    "end_time": "2024-06-12T09:14:27.983Z",
    "total_bytes_billed": 21474836480,
    "project_id": "emea-analytics-pd",
    "dataset_id": "retail_sales"
}
```

By querying the Monitoring dataset, it was possible to reconstruct detailed usage patterns: which teams or users consumed the most resources, which queries drove the highest costs, and how usage evolved over time.
Additionally, since the data resides in BigQuery, it can be analyzed using standard SQL queries, integrated into dashboards, or further processed by machine learning models for anomaly detection and forecasting.

3.1.5 Industrialisation

Industrialisation in the context of cloud data platforms refers to transforming what is manually created, resources and instructions, into repeatable, scalable, and automated practices. When applied to BigQuery, industrialisation ensures that resources such as datasets, tables, scheduled queries, and permissions are no longer created by hand through the Google Cloud Console, but are instead defined and managed consistently through Infrastructure as Code (IaC). Terraform plays a central role in this process. By writing declarative configuration files, every resource in BigQuery can be described as code.

In practical terms, during the Data Exploration Dashboard project, individual components were first tested in a sandbox environment, where queries and tables could be refined experimentally. Once validated, these components were transitioned to production by embedding them into Terraform modules. For example, I extended the existing Monitoring project by adding a dedicated module named storage_dataex. This module encapsulated all resources required for the dashboard pipeline: the scheduled query for metadata extraction, the creation of destination datasets and tables, service account provisioning, and Identity and Access Management (IAM) configuration.
The outcome was a self-service infrastructure model: reusable Terraform modules allow other teams to provision the same type of resources with minimal effort, while still respecting organizational rules.

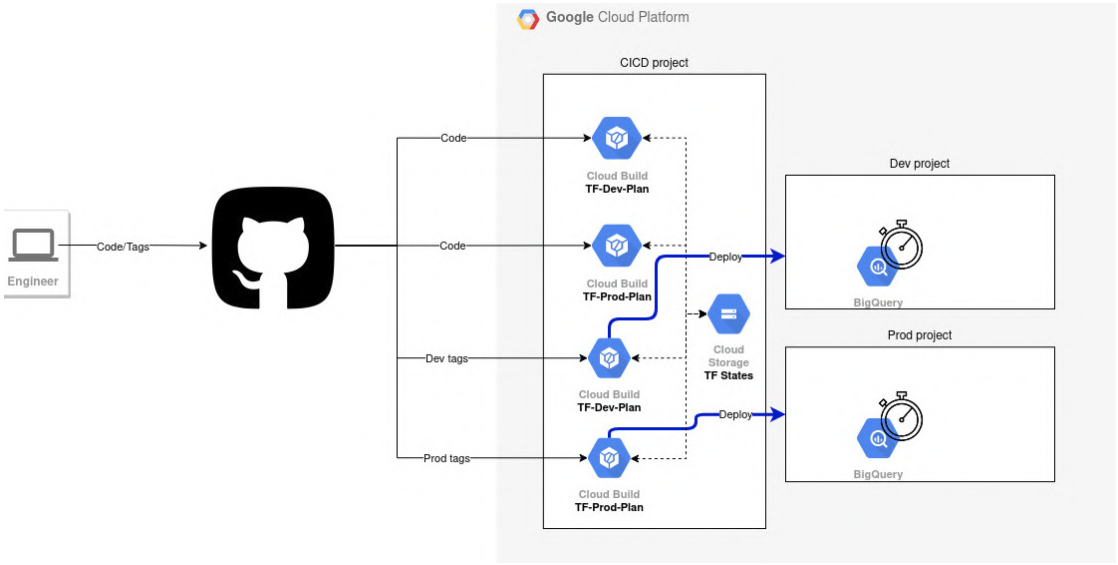


Figure 3.1: Industrialisation using GitHub as a repository to create BigQuery resources

The diagram above illustrates the architectural flow of how Terraform is integrated into a CI/CD pipeline in order to industrialize the management of BigQuery resources.
At its foundation, the process begins with the engineer, who develops the necessary Terraform code locally and commits it to GitHub. This central repository function as a version control system, as the backbone of collaboration, where every modification to the infrastructure can be tracked, reviewed, and approved. Once the code is pushed, the workflow branches depending on the type of change and the target environment. Code tagged for development initiates a dedicated pipeline, while production tags trigger a separate and more controlled process. In both cases, the pipeline is orchestrated through Google Cloud Build, which generates a Terraform plan before any deployment is carried out. This planning phase is crucial, since it provides full visibility into the proposed modifications, allowing engineers and reviewers to verify their correctness and impact before applying them to the infrastructure.
The Terraform state is stored in Cloud Storage, a central repository that records the current status of all managed resources. This state file plays a critical role in ensuring idempotency and consistency, as it prevents duplicated resources or conflicting changes when multiple engineers work simultaneously on the same environment. By maintaining this persistent record, Terraform can guarantee that running the same code repeatedly will always produce the same infrastructure.
After validation, the changes are applied to the appropriate Google Cloud projects. The development project serves as a sandbox, where new features and modifications can be tested safely, while the production project is reserved for stable deployments that support business-critical operations. In this particular case, BigQuery stands at the center of the managed resources, but the same approach extends seamlessly to related components such as datasets, scheduled queries, IAM configurations, and logging sinks. Overall, the process enhances governance and compliance, as every infrastructure change is recorded, auditable, and reversible.

A representative example of industrialization through Terraform is the automation of scheduled queries in BigQuery using the google_bigquery_data_transfer_config resource, customized under the name scheduled_query.

```
1 resource "google_bigquery_data_transfer_config" "scheduled_query"
     {
2   for_each = local.scheduled_queries
3
4   data_source_id = "scheduled_query"
5   location       = var.location
6   display_name   = each.value.name
7   schedule       = each.value.schedule
8
9   dynamic "schedule_options" {
```

```
10      for_each = length(each.value.schedule_options) == 0 ? [] : ["
        do"]
11      content {
12        start_time = each.value.schedule_options.start_time
13      }
14    }
15
16    service_account_name = google_service_account.query_executor.
        email
17
18    params = {
19      query = templatefile("${path.module}/../../../sql/
        scheduled_queries/${each.key}.sql", {
20        env                = lower(var.env),
21        project_id         = lower(var.project_id),
22        location           = lower(var.location),
23        dataex_project_ids = join(",", var.dataex_project_ids)
24      })
25    }
26
27    depends\_on = [google_bigquery_dataset.dataex_datasets,
        google_bigquery_table.dataex_tables]
28  }
```

The for_each loop allows the configuration to scale effortlessly, enabling multiple queries to be provisioned at once, each defined by a unique key and its associated metadata stored in configuration files. This simple construct immediately introduces modularity: instead of hardcoding queries one by one, they are described as data objects, which Terraform interprets and translates into infrastructure.
Each field in the configuration corresponds to a critical technical aspect. The data_source_id specifies that the transfer is a scheduled query, while the location enforces data residency requirements, in this case within the European region, aligning with GDPR compliance. The display_name and schedule parameters formalize both identification and execution frequency, ensuring that queries run predictably without relying on manual triggers. Through the optional schedule_options block, more fine-grained controls such as start times can be applied conditionally.
One of the most important technical features is the use of a service account. By assigning the query execution to a dedicated service account (service_account_name), queries run securely under a non-human identity with only the permissions required, reducing security risks and allowing IAM policies to be audited.
The params section highlights that queries are externalized into separate SQL files and passed into the Terraform configuration dynamically using templatefile. This decouples infrastructure definition from business logic, allowing SQL analysts to maintain and evolve the query logic independently of infrastructure engineers.

Furthermore, environment-specific variables such as project_id, location, and dataex_project_ids are parameterized, enabling the same Terraform module to be reused across development and production environments with minimal changes.
Finally, the depends_on directive ensures proper orchestration. BigQuery datasets and tables must exist before the scheduled queries can be provisioned. By explicitly encoding this dependency, Terraform prevents race conditions or partial deployments that could otherwise occur in a manual workflow. Once stored in a GitHub repository, every change to a scheduled query—whether a modified SQL statement, a new execution frequency, or a new dataset dependency—can be peer-reviewed and approved through pull requests. Policy enforcement tools such as Terraform Cloud or OPA (Open Policy Agent) can validate configurations against organizational rules before they are deployed.

We can use the same principle to explain the resources created during the extraction of information regarding users' use of BigQuery. The external process has already been discussed, so we will focus on the fundamental resource in Terraform. The Monitoring project relies on the google_logging_folder_sink Terraform resource to capture logs from all projects within a folder and export them into a BigQuery dataset for centralized analysis. The following snippet illustrates a simplified version of the implementation:

```
1   resource "google_logging_folder_sink" "bigquery_usage_sink" {
2   name        = "bq-usage-logs-sink"
3   folder      = var.folder_id
4   destination = "bigquery.googleapis.com/projects/${var.
    project_id}/datasets/${var.dataset_id}"
5
6   filter = <<EOT
7       protoPayload.serviceName="bigquery.googleapis.com"
8       protoPayload.methodName="jobservice.jobcompleted"
9   EOT
10
11  include_children = true
12  }
```

In this configuration, the sink name identifies the resource. The folder argument specifies the target folder, ensuring that all logs from projects contained within that folder are captured. The destination parameter defines the BigQuery dataset into which the logs are exported, using the project and dataset identifiers provided as variables.
The filter section plays a crucial role in selecting only the relevant log entries. In this case, it restricts the export to events originating from the BigQuery service (bigquery.googleapis.com), and specifically to job completion events (jobservice.jobcompleted). Finally, the include_children flag ensures that logs from all

nested projects under the specified folder are also collected.
Once deployed, the sink streams logs continuously into the designated BigQuery dataset, where they become available for structured querying and downstream integration with the dashboard.

3.2 Dataplex for Data Gouvernance

3.2.1 Motivations

During the early stages of my internship, one of the challenges that quickly became apparent was the lack of a unified framework for data governance across the organization. Up until that point, much of the responsibility for cataloging, monitoring, and evaluating data quality had been managed directly and manually by the Data Quality team. While their work was essential, the absence of an integrated platform meant that governance practices were fragmented, processes were not standardized, and scalability remained a critical limitation.

It was within this context that I had the opportunity to engage with an exploratory project based on Dataplex, Google Cloud's native solution for data governance and management, never used before by L'Oréal. This proof of concept was conceived as a way to investigate whether Dataplex could centralize and automate governance practices, reducing the dependency on manual interventions while also aligning with the broader data strategy of the company.
A proof of concept (PoC), is an inchoate realization of a certain idea or method in order to demonstrate its feasibility or viability. A proof of concept is usually small and may or may not be complete, but aims to demonstrate in principle that the concept has practical potential without needing to fully develop it. Actually, given the nature of the project, it can also considered as a proof of value (PoV), it differs by focusing more on demonstrating the potential customer use case and value.

The project offered a valuable chance to connect academic concepts I had studied—particularly around database management, metadata, and governance—with real-world enterprise challenges. Beyond the technical exploration, it also allowed me to collaborate closely with the Data Quality team, specifically in activities related to data cataloging and metadata organization. The outcome of this collaboration was a first presentation of the tool, both at a high-level overview for the data engineering team and later in greater depth for my immediate colleagues.

As part of this proof of concept, I focused on a specific use case known as "Credit", where I integrated the project into Dataplex with the goal of testing how the platform could support data mesh structures, tiered data management, and policy enforcement. The exercise was not only an experiment in technology adoption but also an opportunity to reflect on whether a migration of existing governance tools—some of which are currently managed through platforms like

Collibra[1]—would be feasible within the unified Google Cloud ecosystem.

3.2.2 What is Dataplex?

Dataplex can be understood as an intelligent data fabric designed to simplify the way organizations manage and govern their data while unlocking advanced analytical capabilities. Rather than requiring that all data be physically moved into a single repository, Dataplex works by logically unifying distributed data sources and providing a single layer of organization and control. This means that even if information resides in different projects, systems, or storage solutions, it can be discovered, cataloged, and managed as though it were part of a coherent whole.
A central aspect of Dataplex is its ability to automatically discover data and enrich its metadata, integrating both technical descriptions and business context. This process of logical unification allows teams across the organization to work with a shared understanding of what the data represents, without the need to migrate it from its original location.
Governance, which has traditionally been a complex and fragmented responsibility, becomes centralized within Dataplex. Policies regarding access control, data retention, and classification can be defined once and consistently applied across multiple datasets and environments. In this way, monitoring, authorization, and auditing are not just simplified but also standardized, which strengthens the overall security posture and compliance of the organization.
Dataplex introduces intelligent data management capabilities that elevate it above traditional cataloging tools. It integrates AI-driven features to classify data, assess its quality, and trace its lineage across systems. These functionalities provide transparency over how data is created, transformed, and consumed, while also identifying potential issues such as incomplete or inconsistent records.
In recent years, organizations have increasingly recognized the limitations of traditional centralized data architectures and have begun to move towards more distributed approaches such as the data mesh. Conventional architectures typically rely on a monolithic platform managed by a single central team. Of course this model can provide uniformity in the early stages, but it tends to create silos where data becomes fragmented and duplicated, leading to inconsistency and inefficiencies in both access and use. Furthermore, in centralized systems, accountability for governance, quality, and privacy often remains ambiguous, as responsibilities are not clearly assigned to the producers or owners of the data. This absence of

[1]Collibra is a Belgian software company active in data intelligence. It was founded in June 2008, focused on an application-oriented research into business semantics management and semantic technology at the Vrije Universiteit Brussel. In 2018, Collibra raised $100 million in Series e-financing at a valuation of over $1 billion.

well-defined stewardship undermines trust and hinders effective data governance practices.
Another critical weakness of centralized architectures lies in their limited capacity for scalability and democratization. As organizations grow and business needs evolve, the volume, variety, and velocity of data often outpace the ability of a single centralized team to manage and govern it effectively. At the same time, the demand for data-driven insights is no longer restricted to technical specialists but extends across all business units.

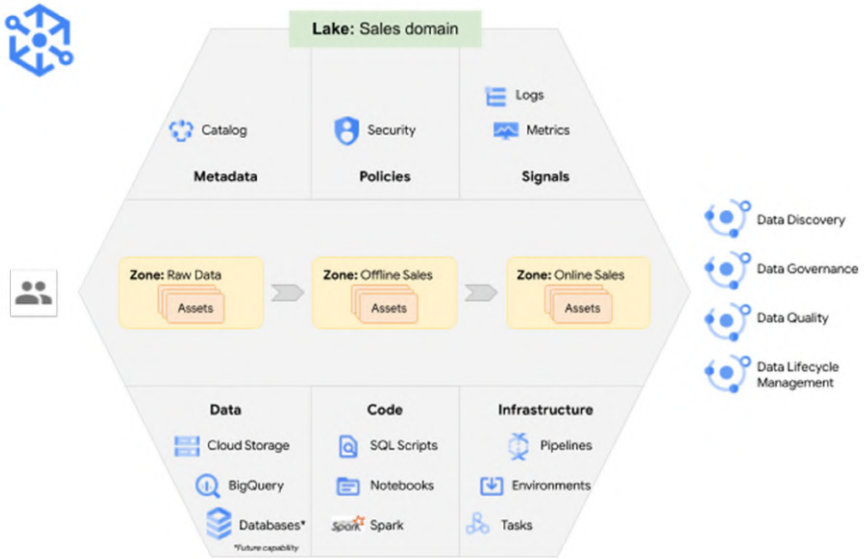


Figure 3.2: Data Mesh in Dataplex

The data mesh paradigm addresses these shortcomings by decentralizing ownership and management. As shown in the figure above, instead of treating data as a by-product of business processes, the data mesh conceptualizes it as a product in its own right, organized along business domains such as marketing, sales, or customer service. In this model, the teams that generate data become its stewards, responsible for its documentation, quality, and accessibility. This decentralization fosters accountability, ensures that governance rules are domain-specific, and enables a more responsive alignment between data assets and the evolving requirements of the business.
Dataplex introduces the concept of a Lake, which functions as the logical container for all assets belonging to a given domain. Unlike traditional data lakes, which often consolidate all data into a single physical repository, a Dataplex Lake unifies distributed assets without requiring physical relocation. For example, Cloud Storage buckets and BigQuery datasets can be associated with the same Lake,

while also incorporating related artifacts such as notebooks, logs, or transformation code. This logical organization supports both analytical and operational workloads, spanning structured and unstructured data, while maintaining a domain-oriented architecture [12].

3.2.3 A Data Mesh Use Case

A further capability is data tiering, which provides a systematic method to manage data based on its maturity and readiness for exposure to different teams. Through tiering, data can be classified into levels—such as raw, curated, or published—each with corresponding policies for access and usage. This ensures that resource allocation is efficient and that sensitive or incomplete data is not prematurely consumed by business users.

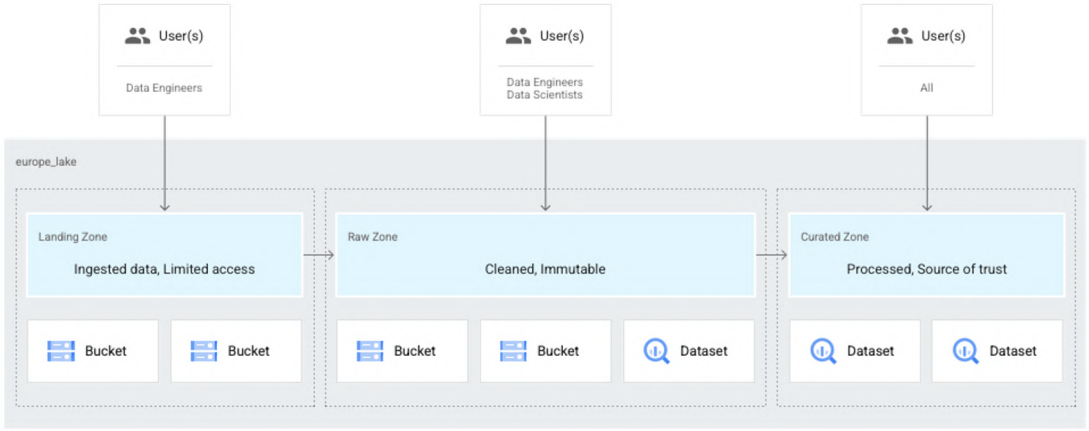


Figure 3.3: Data tiering in Dataplex

As already mentioned, we applied the data mesh approach to a use case project, internally referred to as "Credit". Shortly, this project is related to carbon credit management, in the broader context of L'Oréal's sustainability initiatives. However its nature was not central to our experiment, since we were more interested in exploiting a concrete framework in which to test Dataplex.
The credit use case was organized according to three logical zones, or tiers. The first of these, the Landing Zone, functioned as the entry point for all incoming data. Here, raw files arriving from Google Cloud Storage and tabular from BigQuery were collected in their original form, without preprocessing. This layer is comparable to gathering ingredients before cooking: while all the necessary elements are present, they are not yet suitable for direct consumption or use. In practice, the Landing Zone provides a space where unrefined, heterogeneous data could be

securely ingested and stored without immediate governance constraints beyond access control.
The next step in the pipeline was the Staging Zone, where the emphasis shifted from ingestion to transformation. In this environment, the raw datasets collected earlier were standardized and reshaped to make them more coherent and usable. Continuing the culinary metaphor, this phase was akin to preparing the ingredients—washing, cutting, and organizing them—so that they could later be assembled into a final dish. By applying cleansing and structuring processes, the Staging Zone allowed data engineers and analysts to work with datasets that were consistent, organized, and better suited for exploratory analyses or intermediate business needs.
Finally, the Exposition Zone represented the curated layer of the architecture. This was the environment in which data reached its highest level of readiness, optimized for end users and practical applications. Datasets at this tier had undergone the necessary processing and validation, ensuring that the information could be used directly for reporting, analytics, or more advanced tasks such as training machine learning models. To return to the metaphor, the Exposition Zone is where the dish is complete and ready to be served: refined, reliable, and adapted to its final purpose.

3.2.4 Governance and Lineage

Governance in this context should not be seen as a static layer of control, but rather as a dynamic framework that ensures reliability, accountability, and transparency in the way organizations interact with their data assets.
Traditional approaches to governance often relied on manual oversight and fragmented metadata repositories, which proved insufficient in environments where data is generated across multiple systems and domains. Dataplex, by contrast, provides a unified approach that scales seamlessly with organizational complexity and integrates directly with cloud-native services.

A central element of this framework is the Dataplex Universal Catalog, which functions as the connective tissue of governance by consolidating metadata from diverse data sources into a single, intelligent view. Through automated data discovery, organizations can continuously scan distributed storage systems such as Cloud Storage or BigQuery to uncover and classify both structured and unstructured assets. This capability mitigates the risk of data sprawl, allowing that datasets, once hidden in silos, to be brought into a coherent structure where they can be profiled, validated, and made accessible under controlled policies. The catalog also supports advanced data profiling and quality assessment, two functions that are increasingly indispensable in environments where decisions depend on reliable and traceable information. Profiling provides statistical measures such as value

distributions, for example common patterns, or null counts within datasets, allowing teams to detect anomalies or evaluate compliance with internal standards. Coupled with quality validation rules, these mechanisms act as safeguards against the propagation of incomplete, inconsistent, or inaccurate data. In practice, quality checks are embedded into the lifecycle of the dataset itself.

Dataplex further enhances governance by fostering semantic clarity through glossaries and tagging mechanisms. Technical metadata alone is insufficient if business users cannot interpret its meaning or relevance. To address this, Dataplex provides tools for defining shared business glossaries where terms, definitions, and policies are codified and consistently applied across datasets. Tags and labels, in turn, create an additional organizational layer that allows assets to be grouped by characteristics such as sensitivity, lineage, or domain relevance.

An additional advantage of Dataplex lies in its ability to enforce centralized security and access controls without disrupting the autonomy of business domains. While data remains logically organized into lakes and zones, policy enforcement is managed from a unified governance layer. This ensures that authorization, retention, and classification policies are consistently applied across the organization, regardless of where the data physically resides. For instance, a dataset stored in Cloud Storage or BigQuery inherits the same governance rules when cataloged in Dataplex, reducing the risk of misalignment between security practices and business needs.

In this sense, in comparison with Collibra, Dataplex does not just replicate what other governance platforms, but complements and extends them with a cloud-native approach tailored to distributed architectures. Its integration with BigQuery, Vertex AI, Pub/Sub, and other GCP services anchors governance directly within the operational and analytical workflows, bridging the gap between abstract governance frameworks and their practical implementation. The potential of Dataplex for governance lies in its ability to harmonize three dimensions that were historically fragmented: metadata unification, policy enforcement, and semantic interpretation. By embedding these capabilities into a single platform, organizations can transform governance from a compliance-driven obligation into a foundational enabler of trust and innovation across their data ecosystems.

By leveraging lineage tools in Dataplex and BigQuery, organizations gain a transparent map of their data flows. In the context of the credit use case, the diagram below illustrates the concept of data lineage in a intuitive way. The root element, Credit_Dim_v1, represents the initial dataset. From this source, multiple transformation processes are triggered, which eventually feed into downstream views for different receivers, such as VW_Credit_Clients and VW_Credit_Managers. Each transformation step is captured as a node in the lineage graph, creating a complete trace of how data evolves from its raw form into business-ready outputs.

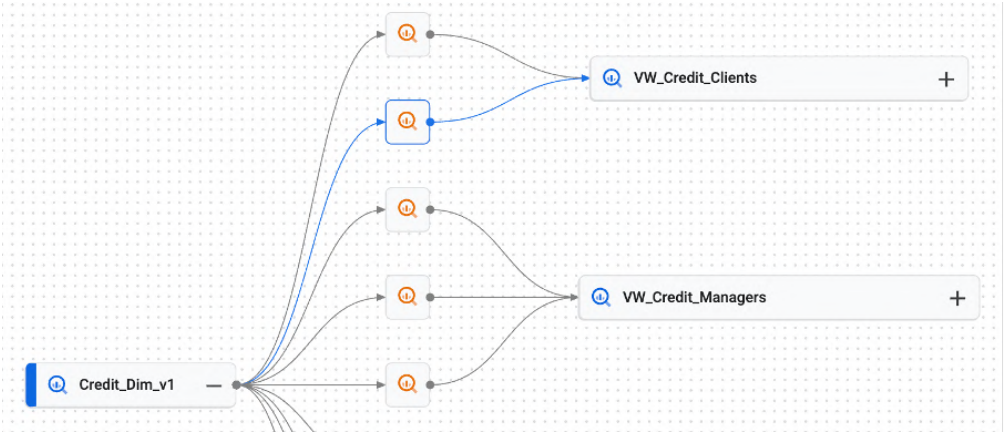


Figure 3.4: Example of Data Lineage in Credit use case

From a technical perspective, the lineage model in Google Cloud is structured around three main elements: processes, runs, and events. Each transformation step in the diagram corresponds to a process, which is defined by the type of operation (for example, a BigQuery job). Every execution of that process creates a run, with attributes such as execution time or state. Within each run, specific events capture the precise flow of data from source to target, allowing the lineage graph to be reconstructed with high fidelity.
This offers several advantages. First, lineage provides a traceability mechanism, enabling errors to be traced back to their root causes rather than being addressed superficially. Second, it supports impact analysis, allowing engineers to understand the downstream effects of modifying or deleting a dataset before taking action, thereby reducing the risk of disruptions. Third, it enables data consumers to confidently assess whether their inputs stem from reliable and authoritative sources. This visualization underscores why lineage is such a critical component of modern data governance. In complex ecosystems, data rarely remains static, instead, it undergoes multiple refinements and aggregations as it flows through pipelines.

3.3 CrunchML: A project to speed up column classification

The third project, and perhaps the most ambitious, revolved around column classification using Vertex AI and BigQuery ML, where I sought to apply machine learning models to automate the recognition and classification of data columns within large-scale datasets, addressing challenges of accuracy, efficiency, and adaptability in data management.

3.3.1 Application Context

Before delving into the description of CrunchML, it is important to establish the context by examining the existing project already deployed in production, known as Crunch!. This application, fully hosted on Google Cloud Platform, represents a cornerstone of L'Oréal's data processing pipeline for retail sell-out data.
Its primary role is to receive raw files provided by retailers, clean and prepare them, and subsequently transform this heterogeneous data so that it can be ingested directly into the Sell-Out Europe 360 (SO E 360) platform. The challenge addressed by Crunch! stems from the absence of any uniform standard for these raw inputs. Retailers submit files in disparate structures and formats, creating an urgent need for harmonization before the data can serve as a reliable foundation for analytics and reporting.
To overcome this heterogeneity, L'Oréal has introduced the concept of a Universal File Format (UFF), which functions as a target schema into which all raw retailer files are transformed. By aligning the outputs into this standardized structure, Crunch! ensures that the data feeding into the Sell-Out 360 Data Foundation is both consistent and comparable, thereby providing a trustworthy base layer for business intelligence.
Beyond the purely technical aspects, Crunch! also integrates a dedicated web portal to facilitate the user experience, ensuring that analysts and operational staff can interact with the system in an intuitive and efficient manner. Equally significant is the emphasis on process standardization across the company's operational hubs. Crunch! also embodies an organizational strategy that includes incident management, service request handling, and the definition of Service Level Agreements (SLAs). These measures guarantee that uniform practices are upheld across regions and teams, reducing fragmentation in operations and promoting reliability in data availability.
At its core, Crunch! operates through a library of reusable Python scripts designed to execute the mapping and transformation of retailer data into the UFF. The logic defining which transformations to apply, as well as the sequence in which they are executed, is configured by a human analyst during the onboarding of each new retailer. This configuration is then stored within the application and reused each time files are received, ensuring that the ingestion pipeline runs automatically without requiring repeated manual interventions.

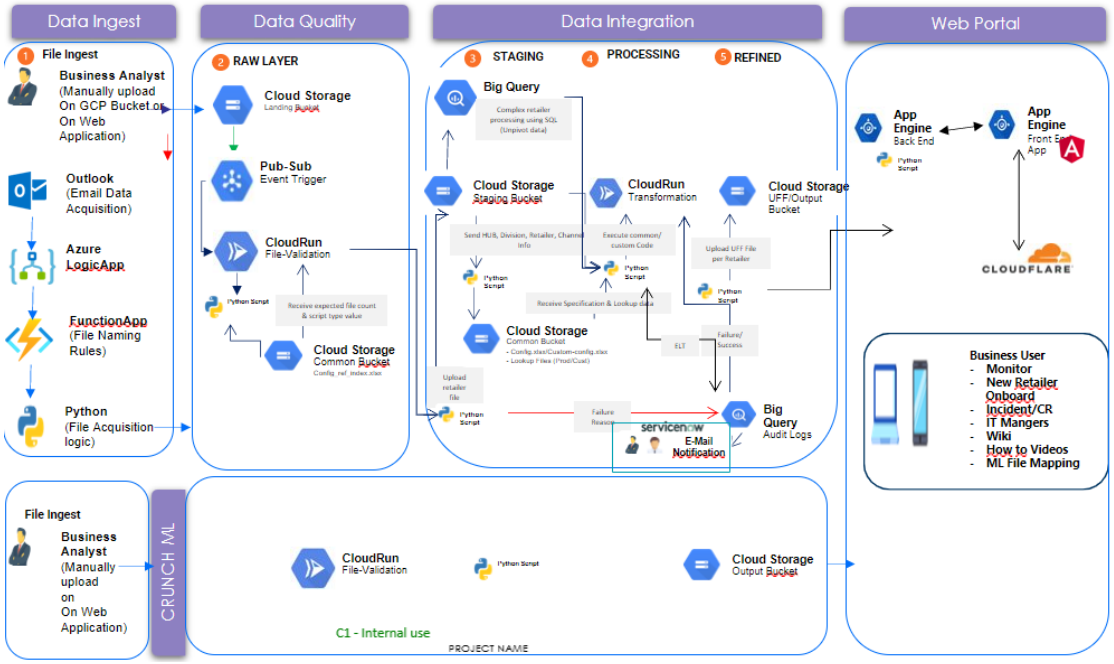


Figure 3.5: Crunch! architecture, including CrunchML at the bottom

At its core, the architecture can be divided into four main domains. The Data Ingest layer is the entry point where retailer files are first acquired. Several ingestion channels coexist here: manual uploads by business analysts to a Google Cloud Storage bucket, email-based acquisition through Outlook, automation pipelines leveraging Azure LogicApps and FunctionApps, and Python scripts designed to enforce file naming rules. This diversity of ingestion methods reflects the heterogeneous and non-standardized nature of the raw data files provided by retailers, which can arrive in varying formats and through multiple channels.

Once ingested, the data enters the Data Quality stage, beginning with its storage in a raw landing bucket. A Pub/Sub trigger mechanism automatically activates validation routines, executed within Cloud Run environments. Python-based validation scripts ensure that the files conform to the expected structure and content rules. Files that fail validation are redirected to a failure bucket, with errors logged and notifications generated, allowing incident management to be handled promptly. Files passing this stage are moved into a common bucket, ready for further processing.

The third domain, Data Integration, represents the staging and transformation phase. Initially, the data is transferred into BigQuery and Cloud Storage staging buckets, where retailer-specific mapping and transformation specifications are retrieved. Python scripts and Cloud Run services perform the required transformations, including both standardized routines and custom code execution when

retailer-specific logic is needed.
The refined output is then stored in the UFF bucket. This stage also features monitoring mechanisms through BigQuery audit logs, ensuring traceability and compliance.
The final layer, the Web Portal, acts as the primary user interface for business stakeholders. Built with a back-end hosted on App Engine and a front-end developed in Angular, the portal provides users with the ability to monitor processes, onboard new retailers, manage incidents, and access resources such as wikis, tutorials, and documentation. Distribution is facilitated through Cloudflare, ensuring performance and scalability across regions.
While this pipeline is already a mature solution, the lower section of the diagram highlights the planned CrunchML extension, which remains under development. This component is envisioned as a machine learning enhancement to automate the mapping source columns from retailer files to target columns in the UFF. It is at the base because it could potentially use machine learning to optimise, and above all automate, the entire flow.
Despite its strengths, the current solution presents two fundamental challenges that highlight its limitations. The first concerns scalability: the manual analysis and construction of retailer-specific configurations is a resource-intensive process, both in terms of time and cost. As the number of retailers grows, this manual approach increasingly becomes a bottleneck. The second issue is the volatility of retailer file formats. Changes in the structure of incoming files often occur with little or no prior notice, leading to failures in the ingestion pipeline and delays in the availability of data within SO E 360. Each such disruption demands additional effort to adapt and maintain the configurations, thereby driving up operational costs and reducing the agility of the overall system.

3.3.2 Functional Requirements

In order to explore the integration of Machine Learning into Crunch!, it was important to start by revisiting the mechanics of how the system currently handles inputs, and then project how automation could simplify these tasks.

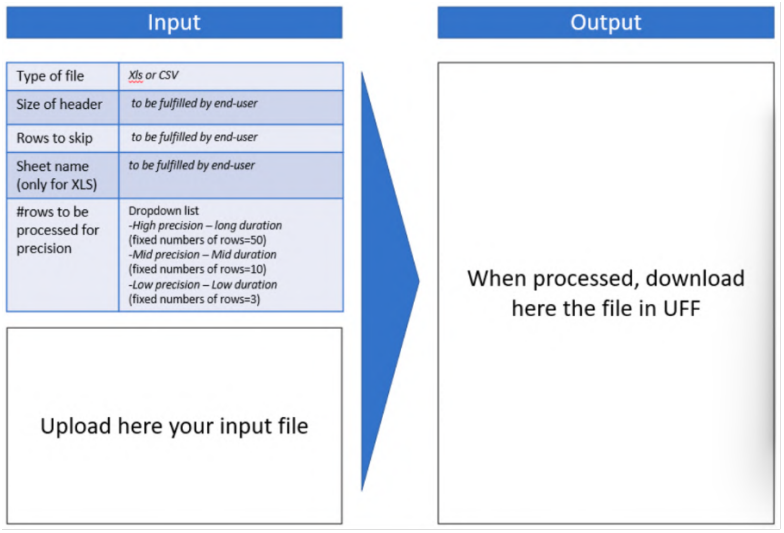


Figure 3.6: Crunch! application interface for feeding an input file

The diagram illustrates the flow from input to output, showing the parameters that must be defined before any file can be processed into the Universal File Format (UFF). As said, at present these parameters are configured manually by the end user. The first input parameter relates to the type of file. Crunch! accepts files in either XLS or CSV format: since there is no common standard across sources, the application must be flexible enough to interpret both file types. The user also specifies the size of the header and the number of rows to skip, parameters that determine which parts of the file contain metadata rather than usable data. These steps are essential to ensure that the system isolates only the relevant dataset for processing, but again, they require manual effort and knowledge of the file's structure. Similarly, when XLS files are uploaded, the sheet name must be provided by the user so the application knows which worksheet to extract data from, as files often contain multiple tabs with varying content.
Another critical input relates to the number of rows to be processed for precision. The system offers three levels of processing detail: high precision, which takes longer and involves up to 50 rows; medium precision, which balances accuracy and performance using 10 rows; and low precision, which is the fastest but limited to 3 rows. This choice reflects a trade-off between processing time and the reliability of column recognition. At present, the user must make this choice manually depending on the characteristics of the file and the expected data quality.
Once the parameters are defined and the input file uploaded, the system processes the data and outputs a standardized file in the UFF format, ready for use in the Sell-out 360 platform. The ultimate vision of introducing Machine Learning is to automate much of what is now manual. The objective of the model is automatically

recognize and map columns to their corresponding targets in the Sell-out 360 foundation tables, drastically reducing the reliance on human analysts. the first step to envisage therefore focuses on training the model with simple files that contain clear headers, have a one-to-one relationship with the target schema, and require no additional transformations. The trained model would then be accessible through an API, with source files uploaded into Google Cloud Storage. By calling the API, Crunch! would receive back a suggested mapping of source to target columns.
Any columns that the model cannot confidently identify would still be flagged for manual intervention, but the goal is to progressively reduce these cases over time as the model learns from more data. In this way, the process shifts from a heavily manual exercise—where the end user configures each detail before uploading—to a more intelligent, automated workflow, improving resilience against changes in file formats.

3.3.3 Implementation

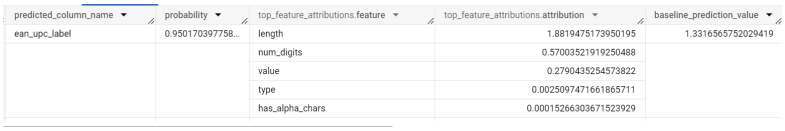
When I first approached the project, the initial task was to identify the most suitable model and input configuration for the data recognition challenge. In fact, before building the entire infrastructure, it is always interesting to understand the feasibility of the task in order to make it repeatable and improvable. A preliminary prototype was developed within BigQueryML, which served as the starting point for the investigation.
BigQuery ML stands as a transformative tool within the realm of data science, fundamentally altering how machine learning and artificial intelligence can be leveraged, particularly on a massive scale. It is an innovation that empowers a broader spectrum of professionals, most notably data analysts, to engage with sophisticated predictive and generative models directly from their existing data warehouse. At its core, BigQueryML allows for the creation and execution of machine learning models using familiar GoogleSQL queries or through the intuitive Google Cloud console, storing these models within BigQuery datasets alongside traditional tables and views. This design elegantly circumvents the traditional bottlenecks associated with large-scale ML, such as the need for extensive programming expertise in specialized frameworks or the cumbersome process of moving vast amounts of data, eliminating the need to move or reformat large datasets for external Python-based frameworks.
BigQuery ML offers a versatile toolkit for working with these models, accessible through various interfaces. Users can interact with models via the Google Cloud console's user interface, the query editor for a SQL-first approach, the bq command-line tool, or the BigQuery REST API, which will be used later for inference and process automation.

Beyond its core machine learning capabilities, BigQuery ML extends its reach into the world of generative AI and pretrained models. It allows users to leverage remote models, which are BigQuery ML models built over Vertex AI models, for a variety of advanced tasks. This includes text generation and embedding, leveraging Vertex AI's powerful text and multimodal models. Furthermore, it incorporates functions to generate specific values using Vertex AI hosted models and to perform time series forecasting using its built-in TimesFM model. The integration also encompasses a range of Cloud AI APIs, enabling natural language processing, machine translation, document processing, audio transcription, and computer vision tasks.
The platform supports a diverse array of models, which are broadly categorized into internally trained and externally trained models. The internally trained models, built directly into BigQuery ML, include algorithms for contribution analysis, linear and logistic regression for prediction and classification, K-means clustering for data segmentation, and matrix factorization for recommendation systems. It also features Principal Component Analysis (PCA) for dimensionality reduction and a sophisticated time series model, for forecasting and anomaly detection. For more complex and specialized needs, BigQuery ML can also interact with externally trained models from Vertex AI. This includes Deep Neural Networks (DNNs) and Wide & Deep models for TensorFlow-based classification and regression, Autoencoders for unsupervised anomaly detection, and models based on XGBoost (Boosted Trees) and Random Forest. It also supports AutoML, a service that automates the building and deployment of classification and regression models on tabular data at a large scale.
To better understand how BigQueryML enables users to create, train, and evaluate machine learning models, using only GoogleSQL queries, here is the query launched to train the first model.

```
1  CREATE OR REPLACE MODEL `emea-sbxowafi-gbl-emea-dv.ml_test.
       semi_harmonized_fields_prediction_prod`
2  OPTIONS(model_type='BOOSTED_TREE_CLASSIFIER',input_label_cols=['
       column_name']) AS
3  SELECT
4      column_name,
5      value,
6      type,
7      length(value) as length,
8      LENGTH(REGEXP_REPLACE(value, r'[^\d]+', '')) AS num_digits,
9      REGEXP_CONTAINS(value, r'[A-Za-z]') AS has_alpha_chars
10 FROM `emea-sbxowafi-gbl-emea-dv.ml_test.training_prod_data`
```

The existing machine learning model, which I have inherited for the purpose of this analysis, was constructed using a Boosted Tree Classifier within the BigQuery ML framework, the most performant model. This particular choice of model is

well-suited for its classification task, and its configuration thoughtfully incorporates a blend of raw data attributes and features derived through engineering. The column to predict is defined in the array input_label_cols, in this case it is column_name. While the foundational features, including value, and type, serve as the primary descriptive inputs, which correspond only to a subclass of all the attributes provided, those chosen are those that were most relevant in training. Beyond these fundamental inputs, the model's predictive power is significantly enhanced by the inclusion of several carefully engineered features. For instance, length was calculated to provide a quantitative measure of the column's length, while num_digits was derived using a regular expression to specifically count the numerical characters within each value. This level of granularity is further enriched by has_alpha_chars, a boolean feature that indicates the presence of alphabetic characters. By integrating these engineered features, the model is able to learn more nuanced patterns and relationships within the data, moving beyond simple categorical or numerical inputs to capture more complex structural information.



predicted_column_name	probability	top_feature_attributions.feature	top_feature_attributions.attribution	baseline_prediction_value
ean_upc_label	0.950170397758...	length	1.8819475173950195	1.3316565752029419
		num_digits	0.57003521919250488	
		value	0.2790435254573822	
		type	0.00250974716618657 11	
		has_alpha_chars	0.00015266303671523929	

Figure 3.7: Example of how some features are more influential in the classification task

Some may ask what principle was used to choose the input columns to give to the model. Well, this is not the empirical result of blind attempts, but as shown in the figure, each column prediction also shows the probability that it is the correct prediction and which columns influenced the classification phase in percentage terms.
The following output provides a compelling illustration of the predictive capabilities of the machine learning model, showcasing the probabilistic nature of its classifications. Each row represents a specific record for which the model has predicted a likely classification, accompanied by a corresponding probability score. This score signifies the model's confidence in its prediction for a given column. For instance, a prediction for a field identified as ean_upc_code is given with a remarkably high confidence of 96.9%.
This output thus serves as a clear demonstration of the model's performance and provides a nuanced view of the prediction process, highlighting its ability to categorize data while simultaneously quantifying the certainty of each classification.

ASIN,EAN,Product Title,Ordered Revenue,Shipped Revenue - % of Total,S		
ean_upc_code : 96.89999999999999%		
,product_lowest_label : 96.0%		
,sold_amount_in_local_currency : 98.5%		
,sold_amount_in_local_currency : 76.1%		
,sold_amount_in_local_currency : 80.0%		
,brand_hierarchy_level_1 : 86.3%		
,sold_quantity_in_units : 76.3%		
,sold_amount_in_local_currency : 76.2%		
,sold_amount_in_local_currency : 88.2%		
,brand_hierarchy_level_1 : 86.3%		
,sold_quantity_in_units : 78.8%		
,sold_quantity_in_units : 79.0%		
,sold_amount_in_local_currency : 94.3%		
,sold_amount_in_local_currency : 80.4%		

Figure 3.8: Output of column prediction

The results, model comparisons and selection, along with a more detailed account of their performance metrics and limitations, will be presented in the subsequent chapter, dedicated to the analysis of results.

At the outset of the project, and given the promising results already obtained with the preliminary model, I decided to concentrate on two complementary directions: the establishment of a robust infrastructure and the refinement of the Python script that managed the ingestion and processing of input files. The progression of activities was aligned with the schedule set by the project manager, which served as both a roadmap and a framework for prioritization.

The first step consisted in deploying the existing processing code within Cloud Run, which allowed the solution to take advantage of Google Cloud's serverless execution environment. This deployment was followed by a series of tests where the script was triggered through API calls. To facilitate adoption and maintainability of the system, I also documented a detailed User Guide for API callers. This document was conceived as a technical manual and as a bridge between developers, analysts, and business stakeholders. Parallel to these steps, the infrastructure was built and formalized using Terraform, already explored with the previous project which allowed the creation of cloud resources to be automated and standardized. A further dimension of the work involved integrating the machine learning lifecycle into the broader architecture. For this purpose, a Vertex AI pipeline was created, designed to periodically retrain and update the existing model. This pipeline ensured that the system would remain adaptable to new data distributions and file formats, reducing the risks associated with model drift. At the same time, an additional mechanism was required to evaluate improvements objectively.

To disentangle the implementation of the first three milestones—the deployment

of the code in Cloud Run, the testing of the script through an API call, and the documentation of usage for external callers—it is necessary to understand the process of containerization, since this forms the basis of any Cloud Run deployment. Cloud Run executes applications that are packaged in containers, meaning that the code, dependencies, and runtime environment must be encapsulated in a portable image. This ensures reproducibility and allows the same application to run across environments, from development to production.
In this project, the containerization process was defined through a Dockerfile, which specifies the sequence of instructions required to build the container image.

```
1 FROM python:3.10.4-slim-buster
2 ENV PORT 8080
3 ENV PYTHONUNBUFFERED True
4 # ENV CONFIG_FILE "config.yaml"
5 COPY . /app
6 WORKDIR /app
7 RUN pip3 install --no-cache-dir -r requirements.txt && pip3 --no-
    cache-dir install gunicorn==20.1.0
8 CMD exec gunicorn -b :$PORT --workers 1 --threads 8 --timeout 0
    main:app
```

The base image used is a lightweight Python distribution (python:3.10.4-slim-buster), chosen to minimize overhead while still providing a stable runtime for the application. Environment variables are declared at the beginning, such as PORT, which is essential for Cloud Run to route traffic correctly, and PYTHONUNBUFFERED, which ensures that output is logged in real time. The application files are then copied into the container under /app, and this directory is set as the working directory. Dependency management is handled by installing the Python requirements listed in requirements.txt. Additionally, the web server Gunicorn is installed and configured as the process manager. Gunicorn is a production-ready WSGI HTTP server that enables the application to handle concurrent requests efficiently. The final line of the Dockerfile specifies the command executed when the container starts, launching Gunicorn to serve the Python application (main:app) and binding it to the designated port.
Once the container is defined, the challenge shifts to automating its build, storage, and deployment. This is where Terraform and Google Cloud Build come into play and it is employed to codify the infrastructure setup. Within Terraform, a google_cloudbuild_trigger resource is defined to automate the build and release of the container. The trigger is connected to a GitHub repository (oa-emea/crunchml), and it is configured to react whenever a new tag is pushed. The build is defined in a cloudbuild.yaml file, which describes the steps to be executed by Google Cloud Build. The build step uses the official Docker builder to create an image from

the Dockerfile and tag it appropriately. Two tags are applied: one based on the Git tag and one labeled as latest. The resulting images are pushed to the Google Artifact Registry, from where they can be deployed seamlessly to Cloud Run. With this setting, at the base of modern DevOps, relieves the developer from manual deployment tasks. Finally, the diagram summarising this classic architecture, used for the CrunchML project, is as follows.

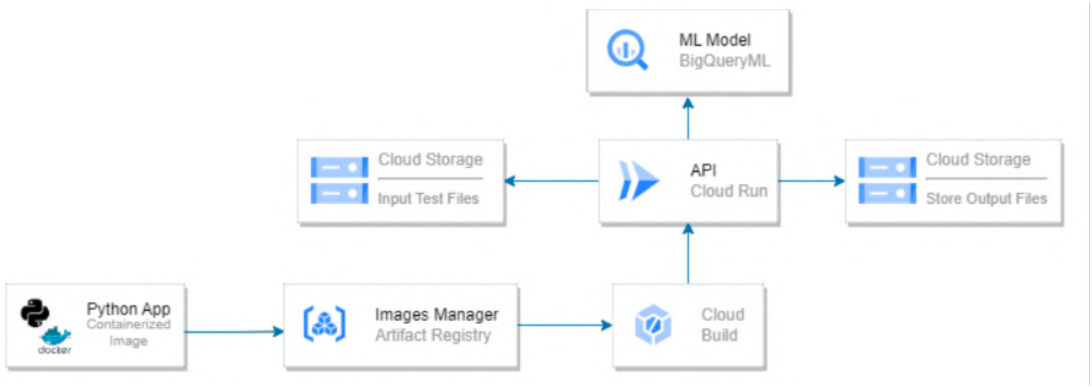


Figure 3.9: CI/CD pipeline for ML model inference: from code to API

To launch testing and provide an initial interface with end users, an initial solution for sending requests to the Cloud Run service was implemented via command line with an HTTP Rest API.

```
curl -X POST \
-H "Content-Type: application/json" \
-H "Authorization: Bearer $(gcloud auth print-identity-token)" \
    https://crunchml-44ywbcq5u-ew.run.app \
-d '{
     "source": "gs://ml_inputs/uki_lpd_boohoo_o_20221106_20221112.
    xlsx",
     "num_rows_header": 1,
     "skiprows": 2,
     "sheet_name": "Sheet1",
     "bucket_destination_name": "ml_inputs",
     "rows_processed": 5
}'
```

This command performs a POST request to the CrunchML service. The request contains headers specifying the content type (application/json) and an authorization token. The authorization mechanism is particularly important, as Cloud Run requires requests to be authenticated. In this case, authentication is handled through an identity token obtained with the command gcloud auth

print-identity-token, which returns a token tied to the authenticated user session in the Google Cloud SDK.
The body of the request contains several critical parameters. The attribute "source" specifies the location of the input file stored in Google Cloud Storage, which serves as the entry point for the data pipeline. The "num_rows_header" parameter defines how many rows in the file correspond to header information, while "skiprows" indicates the number of rows to be excluded from processing at the start of the file, necessary to address the heterogeneity of retailer files, which often contain metadata, comments, or formatting rows before the actual dataset begins. For files in spreadsheet format, the "sheet_name" parameter identifies the specific worksheet to be processed. The "bucket_destination_name" parameter determines the Cloud Storage bucket where the processed output will be saved, and "rows_processed" establishes the number of rows sampled from the input file for processing and subsequent prediction by the BigQueryML model. This parameter plays a crucial role in balancing accuracy and computational efficiency. During testing, it was observed that predictions based on as few as three rows could be obtained very quickly but lacked reliability. Conversely, processing larger samples, such as twenty rows or more, produced far more accurate predictions, though at the expense of higher latency.

3.3.4 Improvements and MLOps

To evaluate the improvements made to the Python application, particular attention should be given to its interaction with BigQuery, as this component constitutes the computational core of the system. The prediction workflow was designed around the google.cloud.bigquery client library, which allows Python applications to interact directly with BigQuery through SQL queries. An illustrative fragment of the improved code is provided below:

```
query = (
    "SELECT * FROM ML.PREDICT(MODEL `PROJECT_ID.ml_test.
   ml_prediction`, "
    "(SELECT * FROM (" + query + ")))"
)
# Calling the query
job = client.query(query)
# Converting the results into a DataFrame
pred_df = job.to_dataframe()
```

This snippet demonstrates the use of the BigQuery ML prediction function. A SQL query string is constructed that calls a pre-trained model (PROJECT_ID. ml_test. ml_prediction) and applies it to the input data. The query is executed

through the BigQuery client, and the resulting prediction job is finally converted into a Pandas DataFrame, which provides a convenient format for subsequent processing and integration into the pipeline.
Before these improvements, the original application relied on a highly inefficient strategy, where predictions were requested individually for each row and column combination. To illustrate, when processing a modest batch of five rows, the script would send separate requests for each column. With an average of fourteen columns per file, this required around seventy connections to BigQuery. Such an approach was not only computationally expensive but also unsustainable in terms of data transfer.
Considering that each request had an average size of 60 MB, the total data usage per batch of five rows reached approximately 4.1 GB. Scaling this procedure up for larger batches—for instance, twenty rows—quadrupled the resource consumption, leading to an overhead of roughly 16.4 GB for a single prediction task. To address this inefficiency, I restructured the prediction logic to aggregate queries, exploiting BigQuery's ability to handle multiple rows simultaneously. This was achieved through the UNION ALL SQL function, which allowed inputs to be combined into a single query, thereby reducing the interaction with BigQuery to just one request per batch. This improvement radically optimized both cost and runtime performance. Through systematic testing, I observed that the computational effort on BigQuery's side remained essentially the same regardless of whether three or twenty rows were processed in one batch.
Beyond query optimization, I also improved the robustness and usability of the application. Several input-handling bugs were corrected, ensuring that diverse file formats and layouts could be reliably processed. In addition, the codebase was refactored to eliminate redundant functions, replacing them with faster and more streamlined alternatives. Finally, the prediction output was refined to display only the most likely class in a standardized format, reducing noise and enhancing interpretability for end users.

After completing extensive testing of the functional application within the sandbox environment, I identified the need to extend the project into an independent, production-oriented initiative.
I formally submitted a resource allocation request to my direct manager. This request sought the establishment of a new Google Cloud Platform billing project dedicated to the CrunchML initiative, which until that stage had remained a proof-of-concept confined to a sandbox environment.
A central component of this new phase was to improve and adapt the configuration of a Continuous Integration/Continuous Deployment (CI/CD) framework. So I configured two distinct triggers in Google Cloud Build, both defined using Terraform and YAML files for workflow specification. The first trigger was designed to activate upon the creation of a pull request targeting the primary GitHub branch. Once

triggered, Cloud Build executed a terraform plan command, which generated a detailed summary of the proposed infrastructure modifications. This output included additions, deletions, and changes to existing resources, thereby allowing developers and reviewers to validate the impact of their changes before they were merged. The second trigger was associated with the production branch of the repository and was designed to engage upon the occurrence of a push event. In this case, Cloud Build executed a terraform apply command, which directly applied the approved infrastructure modifications to the production environment.

The final objective was to transition towards the use of Vertex AI, Google Cloud's next-generation machine learning environment. Vertex AI represents a comprehensive platform designed to support the entire machine learning lifecycle, from data preparation and training to deployment, monitoring, and governance. Its purpose-built tools are intended to automate and standardize repetitive processes, thereby allowing data scientists and machine learning engineers to concentrate on innovation and model development rather than on maintenance tasks. By dynamically managing computational resources, the platform reduces both the cost and time associated with model training and evaluation. Furthermore, the introduction of Vertex AI Pipelines has been a key enabler of this shift. Pipelines simplify the creation, orchestration, and execution of complex machine learning workflows by automating sequential steps such as preprocessing, training, validation, and deployment.

At this point, it is pertinent to reflect on the decision to use Vertex AI Pipelines rather than continuing to rely exclusively on BigQuery ML, which was already producing satisfactory results in the early stages of the project. While BigQuery ML remains a powerful tool for quickly prototyping models directly within the data warehouse, it is inherently limited when addressing the broader challenges of model lifecycle management. These challenges are central to the discipline known as MLOps—a methodological framework that adapts the principles of DevOps to the field of machine learning. MLOps integrates people, processes, and platforms in order to maximize business value from machine learning models. From this perspective, the rationale for adopting Vertex AI Pipelines becomes clear. By design, the platform is capable of scaling to support increasingly large datasets and more complex models without jeopardizing performance or reliability, where data evolves over time and models require constant revalidation and retraining. Another key aspect of Vertex AI Pipelines is the facilitation of collaboration. By providing built-in version control of code, data, and configurations, the platform enables teams to work collectively while maintaining transparency and accountability. Each change is tracked, ensuring that the evolution of a model can be audited and understood retrospectively. The capacity of Vertex AI to automate the machine learning lifecycle from end to end—spanning data ingestion, preprocessing, feature engineering, training, evaluation, deployment, and monitoring—represents

a significant improvement over ad hoc solutions.

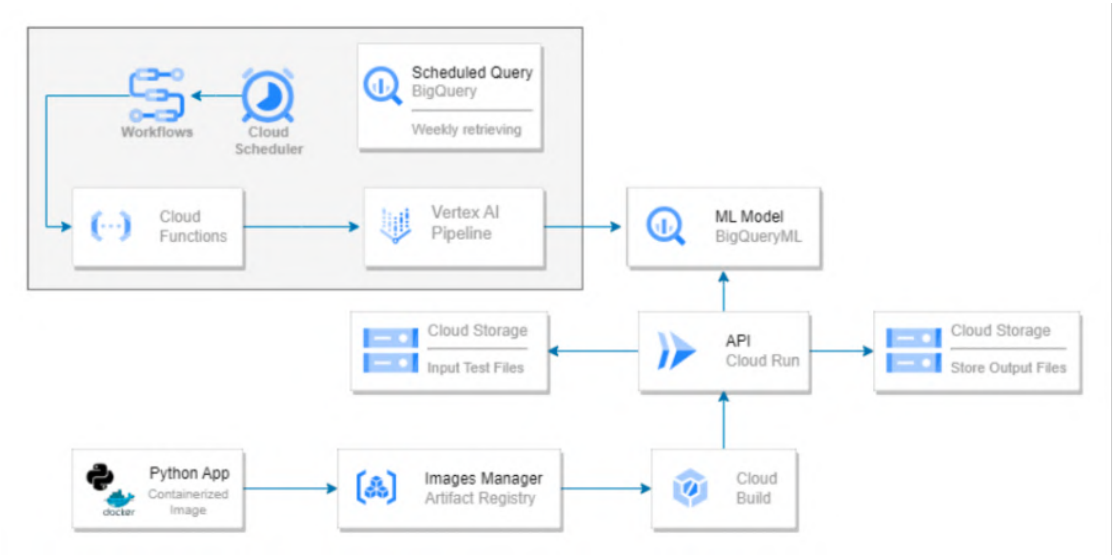


Figure 3.10: The new refined pipeline architecture of CrunchML

At the last stage of the project, I introduced an additional set of utilities to complement the existing infrastructure, as shown in the grey box of picture 3.10 above. This new module, designed specifically for preparing the data required by the upcoming pipeline, was conceived as an intermediate layer within the architecture. The process begins with a scheduled query orchestrated within BigQuery. Initially configured to run on a weekly basis, this query extracts sell-out B2B data from the datafoundation source, which serves as the principal reservoir of structured enterprise data. Given the considerable volume and dimensionality of this dataset, I opted to store the extracted results into a custom table that I designed for this project. This strategic decision was motivated by the need to minimize computational costs. Re-executing complex queries directly on the dynamic datafoundation view would otherwise incur significant resource usage. By centralizing the extracted dataset in a dedicated table, the training pipeline gains direct and consistent access to clean and reliable data, while providing enhanced control over the temporal integration of new records. In parallel, I integrated Cloud Scheduler, which, in combination with Cloud Workflows, provided the orchestration layer for this process. Cloud Scheduler ensured the recurrent triggering of tasks, while Cloud Workflows coordinated the interaction of multiple GCP services and APIs. The workflow itself was defined declaratively using YAML, which delineated each step of the execution sequence. Within this framework, I embedded a Cloud Function that hosted the core logic of the Vertex AI pipeline, written in Python. The pipeline culminates in storing the trained model within both BigQuery and

the Vertex AI Model Registry, thereby providing dual benefits. On the one hand, BigQuery enables direct integration of the model into SQL-based analytics workflows. On the other hand, the Model Registry allows for systematic versioning and comparison against prior iterations, thus fostering iterative improvements. Ultimately, this model becomes accessible to end users via the Cloud Run application, which provides a seamless interface for interaction with the predictions and insights generated by the pipeline.

While designing this system, I also explored alternative approaches, such as the use of Vertex Endpoints for model deployment. Vertex Endpoints allow models trained either within or outside of Vertex AI to be imported and deployed in a managed environment. However, upon weighing the trade-offs, I decided to prioritize Cloud Run due to its serverless nature and cost-efficiency. Unlike Vertex Endpoints, which incur costs even during idle periods, Cloud Run charges are consumption-based, aligning better with the proof-of-concept stage of this project where cost control was paramount. An important feature of this environment is the integration of Jupyter Notebooks, which serve as a powerful exploratory interface. They allow for experimentation and prototyping of the pipeline. In this context, the pipeline was implemented using the Kubeflow Pipelines DSL. The following code snippet exemplifies its definition:

```
@dsl.pipeline(
    name="mlops-bqml-crunchml-classifier",
    description="A Batch Pipeline to Train the Columns
   Classification Model",
)
def pipeline(
    create_bq_dataset_query: str,
    create_bq_preprocess_query: str,
    create_bq_model_query: str,
    project: str = PROJECT_ID,
    region: str = REGION,
    location: str = LOCATION
):

from google_cloud_pipeline_components.v1.bigquery import (
    BigqueryCreateModelJobOp, BigqueryEvaluateModelJobOp,
    BigqueryPredictModelJobOp, BigqueryQueryJobOp)
from google_cloud_pipeline_components.v1.wait_gcp_resources import
    \
    WaitGcpResourcesOp

# create the dataset
bq_dataset_op = BigqueryQueryJobOp(query=create_bq_dataset_query,
   project=project,location=location)

# run preprocessing job
```

```
24 bq_preprocess_op = BigqueryQueryJobOp(query=
       create_bq_preprocess_query,project=project,location=location) \
25 .after(bq_dataset_op)
26
27 # create the model
28 bq_model_op = BigqueryCreateModelJobOp(query=create_bq_model_query
       ,project=project,location=location) \
29 .after(bq_preprocess_op)
```

The pipeline is structured around three main sequential tasks: the creation of a BigQuery dataset, the preprocessing of the data, and the subsequent training of a BigQuery ML model. Each stage is defined through SQL queries and executed using dedicated BigQuery operators provided by Google Cloud's pipeline components. Dependencies between tasks are enforced explicitly, ensuring that preprocessing only begins once the dataset has been created, and that model training only starts after preprocessing has concluded. This orchestration transforms what would otherwise be manual.

What makes this approach particularly effective is how seamlessly Google Cloud's ecosystem components interoperate. BigQuery serves both as the environment for running SQL-based transformations and as the platform for model training. Vertex AI Pipelines provides the orchestration layer, ensuring that workflows are modular and scalable.

Chapter 4

Analysis of Results

This chapter presents the analysis of the results obtained through the methodology described in the previous chapter. The focus here is to assess the outcomes of the proposed approach by illustrating the developed dashboards, examining its alignment with the expected objectives, and discussing both the impact of the solution and the limitations encountered.
For the Dataplex component, the evaluation will not emphasize performance metrics, as the work carried out was conceived as a proof of concept. The central aim was to demonstrate its core functionality, which has already been outlined in the preceding chapter. Accordingly, the discussion in this section will concentrate on the way the solution materializes in practice, highlighting the features implemented and how they contribute to the overarching objectives of the study.
In contrast, the analysis of CrunchML will place greater emphasis on performance assessment. The results of the model are presented in detail, including its behavior under different configurations and its comparative effectiveness against other tested approaches.

4.1 Dashboard Results and Evaluation

The first page of the Looker Studio report focuses on the analysis of platform usage. The initial expectation for this dashboard was to provide a dual perspective: on the one hand, a general overview of usage across all Google Cloud Platform tools, and on the other, a more detailed examination of BigQuery, since this represents the most intensively used service and therefore requires a deeper level of analysis. The data displayed in this dashboard is sourced from the monitoring project, where the logs provide information about the specific GCP services being used. In these logs, the field serviceName identifies the service that generated the entry; for example, bigquery.googleapis.com indicates activity associated with BigQuery.

At the beginning of the project, some key indicators were required, which give an immediate overview of the business. These include the total billed bytes per user, shown both for GCP as a whole and specifically for BigQuery. In addition, the dashboard should highlight the distribution of queries across different regions within the data exploration project. This regional breakdown should be accompanied by temporal analysis, displaying the evolution of usage over the course of the last month.

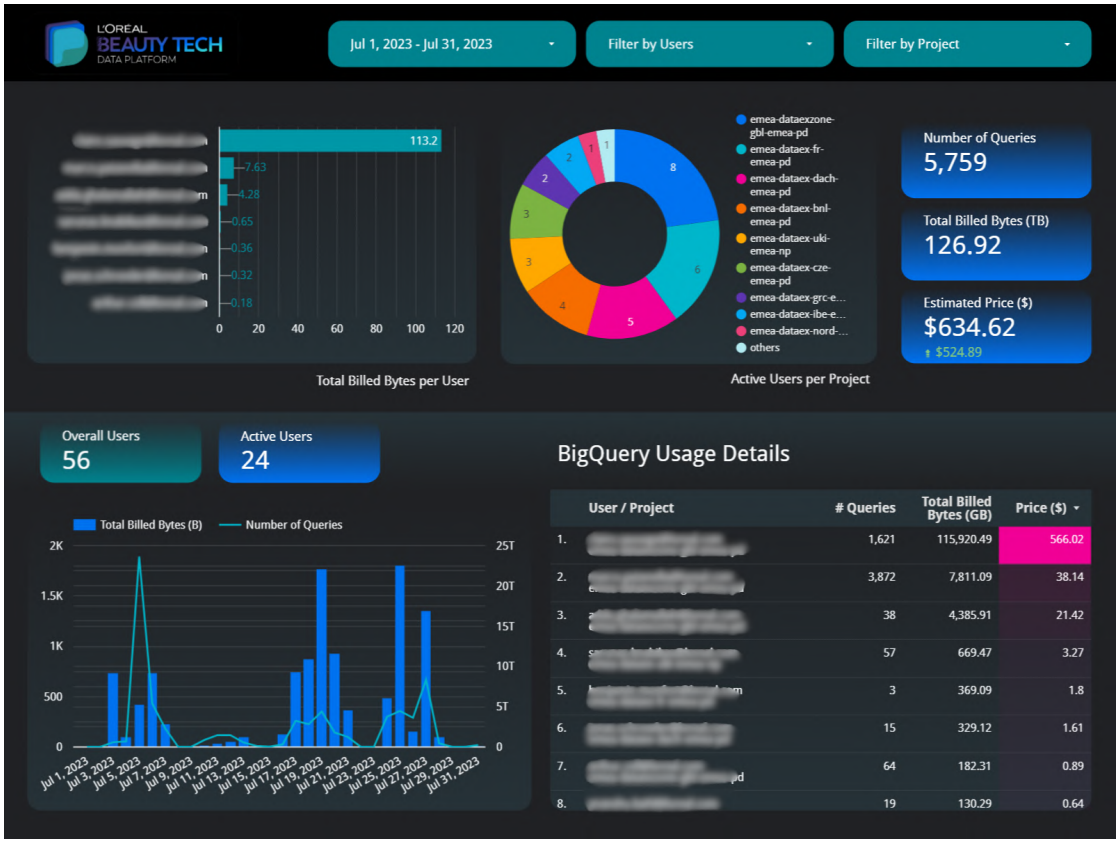


Figure 4.1: Dashboard, first page. Users usage.

The visual structure of the usage dashboard is carefully designed to foreground the dynamics of BigQuery billing and to provide a coherent understanding of the factors driving costs. At the core of the dashboard for usage monitoring, shown in figure 4.1, lies the presentation of total billed bytes, which constitutes the primary determinant of expenditure in BigQuery. By placing this metric at the center of the analysis, the dashboard ensures that users immediately recognize the volume of data processed as the most critical driver of costs.
A distinctive feature of the dashboard is its integration of estimated costs for individual users and projects. This is defined as "estimated" since we do not have the

effective cost associated, from the data extracted, but we could build it over the assumption that the pricing rule of five dollars per terabyte processed, at the moment of development. The final estimation is achieved through a formula that multiplies five by the sum of total billed bytes, normalized by dividing through the power function POWER(1024,4) to convert bytes into terabytes. In other words, the calculation 5 * SUM(auditlog.bigquery.jobCompletedEvent.job.jobStatistics.totalBilledBytes) / POWER(1024,4) provides a sufficient mapping to cost projections.
The temporal dimension of usage is represented in a graph located at the lower left of the dashboard, where a time series plots both total billed bytes and number of queries over the selected period, by default the current month. The decision to display these two variables together arises from the recognition that the number of queries alone can be misleading, as it fails to capture the "heaviness" of queries in terms of bytes processed. By contrast, the joint representation allows a more nuanced analysis, where the relationship between activity volume and data intensity can be observed. The time series therefore provides a more faithful depiction of workload evolution, revealing both spikes of activity and variations in data consumption that would remain hidden in aggregated measures.
The dashboard also includes a donut chart to illustrate the distribution of active users across projects. This visualization offers a rapid overview of relative engagement, usually preferred over pie charts, showing which projects contribute more prominently to the platform's usage. Directly below, a detailed tabular breakdown provides project-specific information at the level of individual users. In order to respect privacy, email addresses are masked, but the table nonetheless retains its analytical function by presenting billed bytes, query counts, and estimated costs. The heat-mapped coloration of the cost column serves as an intuitive signal, highlighting high-cost users or projects and guiding attention toward potential outliers or areas of inefficiency.
An additional element on the top is the feature that allows the user to dynamically select specific projects, individual users, or custom timeframes through a sliding window date filter. Such interactivity is characteristic of modern business intelligence platforms, where dashboards are not static representations but flexible tools that adapt to the analytical focus of the user. In this way, the dashboard supports both exploratory and targeted analyses.
The second page of the report focuses on cost monitoring in relation to predetermined budget allocations, providing a strategic perspective on the financial impact of platform usage. The visual structure is organized to support both aggregated exploration across multiple zones and detailed inspection of individual zones.

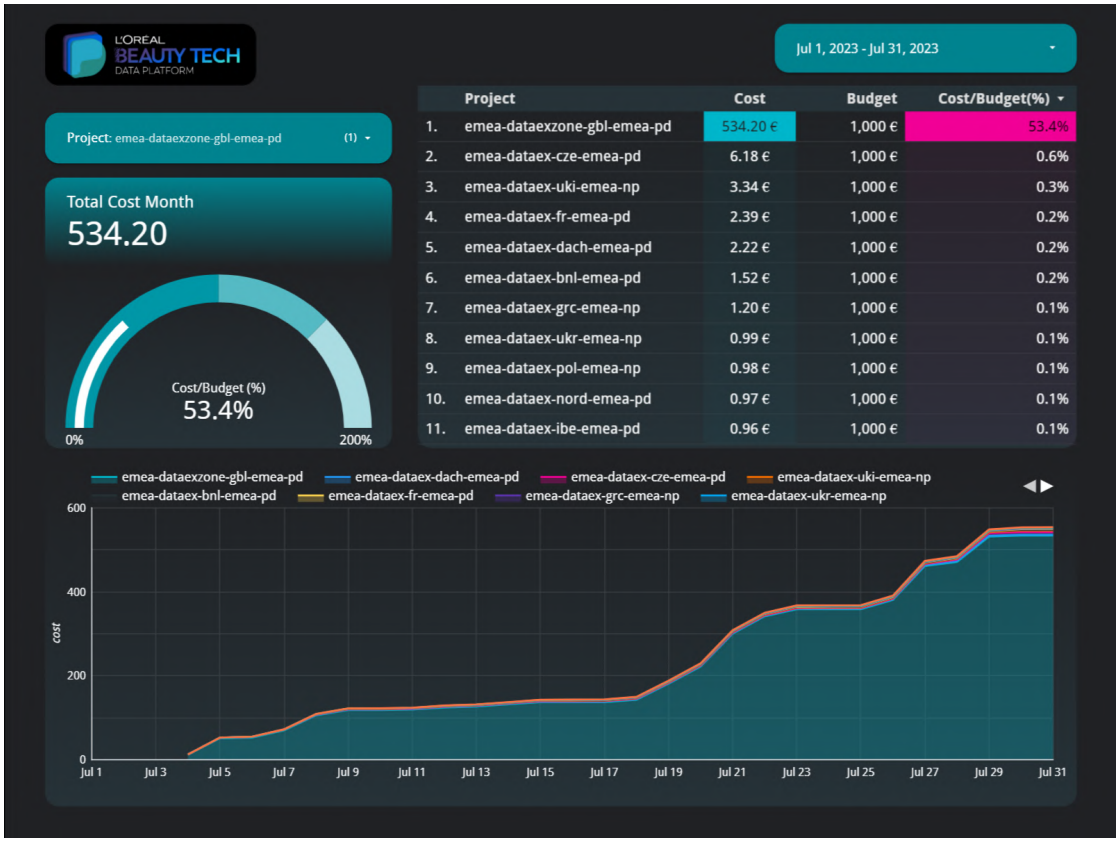


Figure 4.2: Dashboard, second page. Costs.

At the top-right of the dashboard, users can interactively explore data corresponding to different geographic or organizational zones, already mentioned in chapter three. For each zone, the dashboard displays three essential metrics: the total accumulated costs of the projects within the zone, the corresponding budget allocation, and the ratio between cost and budget. This ratio is a key indicator of efficiency, as it immediately reveals the extent to which resources are being consumed in relation to the planned financial limits. The tabular presentation of these values allows users to compare zones directly, highlighting which regions remain comfortably within budgetary thresholds and which approach or exceed critical levels of utilization.
The left section of the dashboard provides a focused view of a single zone selected by the user. It displays the accumulated cost, the assigned budget, and the cost-to-budget ratio in both numerical and graphical form. By doing so, it enables a more targeted assessment, an important feature requested by the business team.
The lower section of the dashboard offers a temporal perspective through a cost trend graph. This visualization plots the cumulative costs across the selected

period, starting from zero on the first day of the month and progressively increasing as usage accumulates. The representation provides an immediate sense of how expenditures evolve over time, revealing the final totals and the pace at which budgets are being consumed. In this way, the time series can anticipate potential overruns before they occur and to link variations in costs to specific operational events.
As well as the previous page, the dashboard is enhanced by interactive functionalities. Users can customize the period of analysis by adjusting the date range, and to focus on specific projects.
As before, the calculation of the cost due to usage is approximate and unfortunately, information regarding billing on GCP is only provided at the end of the month, leaving room only for a forecast for real-time calculations.

Figure 4.3: Dashboard, third page. Storage allocation.

The third page of the report shifts the analytical focus to the underlying storage layer of the platform, leveraging database metadata to provide insights into the size and evolution of datasets. Unlike the previous dashboards, this one does not display information regarding financial costs, since storage volumes have only a marginal

impact on the overall expenditure compared to query execution and computation. Instead, its main purpose is to assess the structural footprint of the datasets and to monitor their progression over time.
The top section of the dashboard provides immediate key indicators in the form of general counters. These summarize the total number of tables, the total number of views, and the overall storage footprint in gigabytes. To complement this overview, a donut chart illustrates the relative distribution of datasets across various projects. Beneath the overview section, the dashboard provides a detailed table listing dataset entities at a more granular level. Each entry contains information about the project and schema of origin, the table name, its type (such as base table or view), the number of rows it contains, and the size it occupies in megabytes. This detailed information allows analysts to identify particularly large or row-intensive tables and to assess how different types of datasets contribute to the overall storage consumption.

Finally, the bottom section of the dashboard introduces a temporal dimension by plotting the number of rows across projects in a time series. This visualization tracks the growth of datasets from a historical perspective, illustrating when significant increases occurred and how data volumes have evolved over time. By relying on row counts as a proxy metric, the dashboard highlights project dynamics without requiring constant recalculation of storage size, which can be subject to compression and formatting variations.

4.2 CrunchML Performance Analysis

Now that the results of the first project have been thoroughly evaluated, the assessment framework for the CrunchML project appears more clearly defined and quantifiable. In order to properly evaluate the outcomes, it is necessary to consider two complementary dimensions. The first concerns the quantitative metrics that determined the best-performing model during training and testing, and the second dimension extends beyond the numerical evaluation of model performance and addresses the practical implications of the system's deployment. Specifically, the effectiveness of CrunchML must also be measured in terms of user experience and system responsiveness. Similarly, usability factors—such as the clarity of outputs, the ease of integration within existing workflows, and the stability of the system under varying loads—represent critical benchmarks for assessing the project's success.

4.2.1 Model Selection

To train and evaluate the model, we employed BigQueryML, conducting a systematic exploration of multiple algorithms in order to identify the optimal performer.

Given the unbalanced nature of the dataset—where certain features, such as the ean_upc_code, are represented by disproportionately large training samples—it was essential to carefully examine performance across a range of evaluation metrics. Relying solely on accuracy in this context could be misleading, as a model might appear to perform well by favoring the majority class while failing to correctly predict less frequent cases. For this reason, we considered a broader set of metrics, including precision, recall, F1 score, and the area under the Receiver Operating Characteristic curve (ROC AUC).
Precision measures the proportion of correctly identified positive predictions relative to all predicted positives, providing insight into the reliability of the model when it classifies an instance as positive. Recall, by contrast, evaluates the proportion of actual positives that were correctly identified, which is particularly critical in unbalanced datasets where the minority class may otherwise be overlooked. The F1 score harmonizes these two metrics by calculating their weighted average, offering a balanced perspective when precision and recall diverge. Accuracy, while still reported, is interpreted with caution, as it can be inflated by the predominance of majority-class predictions. Finally, ROC AUC serves as a threshold-independent measure of separability, capturing the model's ability to distinguish between classes regardless of imbalance.
Log loss, in particular, warrants closer attention. In BigQueryML, this metric quantifies the accuracy of predicted probabilities by penalizing incorrect predictions in proportion to their confidence. Predictions that are both wrong and made with high certainty receive heavier penalties, whereas more cautious predictions are penalized less. Consequently, a lower log loss indicates a model that not only classifies correctly but also assigns reliable probability estimates.

Model	Precision	Recall	Accuracy	F1 Score	Log Loss	ROC AUC
Boosted Tree	**0.919**	**0.941**	**0.930**	**0.924**	**0.185**	**0.996**
Random Forest	0.895	0.920	0.909	0.907	0.213	0.989
DNN	0.873	0.895	0.887	0.884	0.246	0.963
Wide-and-Deep	0.862	0.882	0.874	0.872	0.268	0.958
Logistic Regression	0.842	0.876	0.863	0.859	0.302	0.946

Table 4.1: Performance comparison of BigQuery ML classification models for table column type prediction

The comparative evaluation of the models, presented in Table 4.1, highlights the differences in predictive performance across a range of widely used classifiers. Each algorithm brings distinct methodological characteristics, which in turn shape its performance on the unbalanced dataset under study.
The Boosted Tree model—an ensemble method that builds successive decision trees

where each tree attempts to correct the errors of its predecessor—emerges as the most effective approach across all metrics. Its high precision (0.919) indicates that positive predictions are accurate in the vast majority of cases, thereby minimizing false positives. Equally, its recall (0.941) demonstrates strong sensitivity, meaning that most true positives are successfully captured. Accuracy (0.930) confirms this balanced predictive strength, though it must be interpreted with caution due to the dataset imbalance. More revealing is the F1 score (0.924), which harmonizes precision and recall, indicating the model's ability to control both types of classification error. Notably, the Boosted Tree achieves the lowest log loss (0.185), meaning that its probability estimates are well-calibrated, while its near-perfect ROC AUC (0.996) confirms its excellent discriminatory power across thresholds.
The Random Forest model, another ensemble technique based on aggregating multiple decision trees trained on random subsets of data and features, also delivers strong results. With precision (0.895) and recall (0.920), it approaches the performance of the Boosted Tree but with slightly higher log loss (0.213) and a lower ROC AUC (0.989).
The difference between the two models is that boosted trees reduce bias by sequentially fitting trees to residual errors, producing sharper calibration and higher discriminatory power, while random forests reduce variance through parallel aggregation of decorrelated trees, yielding stability but weaker calibration.
The Deep Neural Network (DNN), inspired by the layered structure of the human brain, relies on nonlinear transformations across multiple hidden layers to capture complex feature interactions. In this experiment, the DNN achieves respectable performance with precision (0.873), recall (0.895), and ROC AUC (0.963). However, its log loss (0.246) is higher, reflecting that while the model classifies well, its probability estimates are less consistent.
The Wide-and-Deep model, which combines a linear component (the "wide" part) with a deep neural network (the "deep" part), is designed to capture both memorization of co-occurrence patterns and generalization across unseen feature combinations. Its performance—precision (0.862), recall (0.882), and F1 score (0.872)—shows that it can model both simple and complex relationships, yet it underperforms compared to purely tree-based methods in this scenario. Its higher log loss (0.268) indicates weaker probability calibration, while its ROC AUC (0.958) shows decent but not exceptional separability.
Finally, the Logistic Regression model, a baseline linear classifier that estimates the probability of class membership by applying the logistic function to a weighted sum of features, unsurprisingly yields the lowest performance across all metrics. With precision (0.842), recall (0.876), and F1 score (0.859), it demonstrates reasonable predictive capacity, but its log loss (0.302) is the highest among the tested models, meaning less calibration. Its ROC AUC (0.946), although respectable, confirms its limited ability to capture nonlinear feature interactions when compared with more

sophisticated classifiers.

Once the best-performing model was identified as the Boosted Tree Classifier, its predictive ability was further assessed through the use of different evaluation metrics, among which the confusion matrix plays a particularly important role.

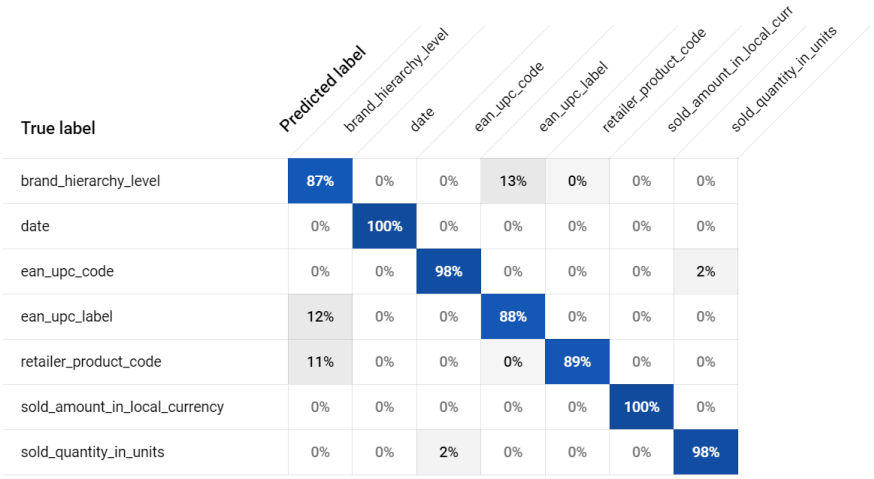


Figure 4.4: Confusion Matrix computed on a validation set using a Boosted Tree Classifier Model

A confusion matrix is a tabular representation that contrasts the true class labels of the data with the labels predicted by the model. Each row of the matrix corresponds to the actual categories, while each column represents the categories predicted by the model. The diagonal elements indicate the proportion of correct classifications for each class, while the off-diagonal elements show misclassifications, thus revealing the overall accuracy and which categories are more prone to errors and "confusion".
In the case presented, the matrix provides a clear overview of how the classifier distinguishes between the different label categories. The high percentages along the diagonal suggest that the model achieves strong performance in most cases. For instance, labels such as date, sold_amount_in_local_currency, and sold_quantity_in_units show prediction accuracies close to or at 100%, indicating that the model is able to recognize them with near-perfect reliability. Similarly, ean_upc_code and retailer_product_code also exhibit high levels of correct classification. However, the matrix also highlights some interesting areas of confusion. For example, brand_hierarchy_level is occasionally misclassified as ean_upc_label, with around 13% of predictions falling into this incorrect category. Conversely, ean_upc_label itself is misclassified as brand_hierarchy_level in about 12% of

the cases, suggesting a reciprocal ambiguity between these two classes. Additionally, a smaller proportion of retailer_product_code instances are misclassified as brand_hierarchy_level, which indicates that while the classifier is generally accurate, these specific labels present overlapping patterns that make them more challenging to differentiate. A marginal misclassification can also be observed between ean_upc_code and sold_quantity_in_units, although this remains minimal compared to the other cases.

4.2.2 Framework Performance

Beyond the intrinsic predictive performance of the models, the responsiveness and operational efficiency of the deployed architecture are equally critical in determining its suitability for production use. In the present setup, the prediction pipeline is exposed through an HTTP GET request routed to a Cloud Run service, which then orchestrates the invocation of a BigQueryML `ML.PREDICT` statement.
Empirical observations suggest that end-to-end latency for a single request averages between 1.2 and 1.8 seconds, with the lower bound representing best-case scenarios under minimal load and the upper bound reflecting background system variability and cold-start effects in Cloud Run. A closer examination reveals that the bulk of this delay—approximately 70–80%—originates from the execution of the BigQuery prediction itself, while the request handling, serialization, and network transfer account for the remainder.
When the system is subjected to moderate concurrency, such as handling 50 to 100 simultaneous requests, latency increases in a near-linear manner, stabilizing at roughly 2.3 seconds on average. Importantly, this increase does not signal bottlenecks in Cloud Run's autoscaling mechanism, which reacts swiftly to load spikes by provisioning additional containers, but rather highlights the inherent query execution time in BigQuery as the dominant factor in responsiveness.
From a cost perspective, the system demonstrates favorable characteristics. BigQuery charges on the basis of data processed, and in the context of `ML.PREDICT`, each query typically scans only a few megabytes, translating into costs well below $0.01 per prediction. Cloud Run, in turn, bills according to the allocated vCPU and memory usage per request duration, with our configuration incurring approximately $0.001 per invocation at standard usage volumes.
Even when scaled to thousands of daily predictions, the combined costs remain modest relative to the operational benefits of serverless infrastructure, particularly the absence of idle-time charges and the seamless scalability.

If we extend the analysis of response times and operational costs to the batch prediction optimization, discussed in the previous chapter, the advantages of restructuring the query execution strategy become evident. In the initial setup, each request was handled individually through Cloud Run, invoking a distinct

call in BigQuery for every row submitted. With an average request for a batch of five rows incurred 4.1 GB of processed data. Scaling this approach linearly to twenty rows resulted in nearly 16.4 GB of processed data for a single prediction task. This overhead was reflected in increased costs—on the order of $0.15–0.20 per batch—and also in longer execution times, averaging around 1.2 seconds per row, which compounded to roughly 24 seconds for the complete set of twenty predictions. Restructuring the prediction logic to aggregate multiple rows into a single query significantly reduced this inefficiency.

Empirical measurements indicated that BigQuery's execution engine handled the aggregated input with nearly identical computational effort as for smaller batches, resulting in a single query runtime of approximately 1.6 seconds for all twenty rows. In terms of costs, the consolidated query processed just 1.1 GB of data, reducing BigQuery charges by nearly 93% compared to the row-wise execution approach. When combined with Cloud Run's billing model—driven by CPU allocation per request—the consolidated method further reduced overhead by minimizing the number of service invocations.

Chapter 5

Conclusion

The work presented in this thesis has demonstrated the potential of combining cloud computing, data engineering, and machine learning to address practical challenges in large-scale data management. The reflections drawn from this research point toward the reality that digital transformation is neither a purely technical exercise nor a finite undertaking, but a continuous process of adaptation, negotiation, and refinement.

5.1 Limitations and Future Development

These considerations are particularly evident in the three projects developed. Conceived and implemented in relatively short timeframes, each of approximately two months, they have already demonstrated their effectiveness in terms of design and performance, thereby proving the feasibility of the approaches adopted. Yet, as is often the case with early-stage implementations, they remain immature and necessarily leave room for improvement. Bugs, usability issues, and limitations in user interaction should not be overlooked, but rather regarded as opportunities for refinement.

The first project, for example, revealed structural constraints, most notably in the ability to anticipate and monitor platform costs in real time. Current reporting mechanisms only provide cost details retrospectively, limiting their value for operational planning. Addressing this challenge through the integration of a predictive machine learning algorithm could not only enhance financial foresight but also evolve in accuracy over time as usage histories accumulate.

Similarly, the dashboard interface, while functional, remains minimal in terms of user experience design, and could evolve into a richer and more dynamic environment with the addition of interactive and user-centered features. Early prototypes are not expected to resolve every challenge but rather to lay the groundwork for

further iterations, informed by empirical testing and organizational feedback. In this sense, the work reaffirms that technological progress in the enterprise setting emerges through cycles of trial, evaluation, and enhancement, where feasibility must first be demonstrated before maturity can be achieved.

The Dataplex project opens yet another dimension to this reflection, as it points to the strategic importance of centralization. While existing tools such as Collibra remain valuable, consolidating data governance within Dataplex could offer a more integrated and sustainable framework for long-term management. Such a migration would inevitably require an upfront investment of time and resources, and careful consideration of the trade-offs involved. However, the long-term benefit of managing data through a unified platform—simplifying oversight, improving interoperability, and reducing fragmentation—suggests that the effort may be justified.

Among the projects undertaken, CrunchML occupies a particular place, both because of the time invested in its development and because of the potential it continues to hold. The work carried out so far has laid the foundation for what could eventually become a production-ready system, as evidenced by the fact that it has already been taken forward by another developer, moving tentatively toward a first beta version.

The limitations encountered are instructive in themselves, as they highlight the distance between a functioning prototype and a system designed for a seamless user interaction.

At present, the workflow begins with a command-line HTTP request, which, while technically compliant with the expected functionality, is far from offering the accessibility demanded in a real-world environment. A more natural interface—enabling users to upload files through drag-and-drop actions and initiate processes with a simple click—remains to be designed. The challenge lies in reducing the friction that separates domain experts from the value that machine learning can deliver.

Another recurring difficulty was the need for users to manually specify parameters such as header size in order to process files in CSV or XLS format. While this method ensures clarity for the algorithm, it imposes a mechanical burden on the user and interrupts the fluidity of the workflow. A promising direction for overcoming this limitation lies in the extension of the Python pipeline to support multiple formats natively, coupled with strategies for automatically detecting the location of relevant information within the file. One possible approach would be to devise pattern-recognition heuristics capable of identifying the true content header. A more ambitious direction would involve incorporating machine learning directly into the preprocessing pipeline, training the system to distinguish between metadata and meaningful data before handing it over to the classifier.

The question of automating this initial stage also opens the door to exploring more advanced forms of artificial intelligence. A retrieval-augmented generation (RAG) system, for instance, could be envisaged as a means of assisting users dynamically.

RAG combines a retrieval mechanism—which searches for relevant information in a dataset—with the generative capacities of large language models, enabling the system to provide context-aware outputs. In the case of CrunchML, a tailored RAG architecture might allow the platform to ingest the uploaded file, interpret its structure, and provide the necessary classification instructions without requiring the user to manually supply input options. Such integration remains speculative at this stage, but it underscores the potential of bringing emerging AI paradigms to bear on practical challenges of usability and automation.

Parallel avenues for development concern the core classification algorithm itself. A system of this kind should not remain static but evolve continuously as new data becomes available. This could involve retraining models on both previously classified datasets and incoming streams, whether in batch or real-time mode. Doing so would increase the robustness of predictions and ensure that the system adapts to the dynamic character of the data it encounters. Realizing this vision might also require leveraging hardware accelerators such as Tensor Processing Units (TPUs), which are specialized processors developed by Google and designed to optimize the training and inference of machine learning models at scale. Likewise, the incorporation of no-code tools offered by cloud platforms could lower barriers to experimentation, allowing faster prototyping and more inclusive collaboration across teams.

In its current form, CrunchML embodies both the promise and the incompleteness of projects conceived within constrained timeframes. It demonstrates feasibility, stimulates further questions, and invites new iterations. Its evolution will depend on the balance between pragmatic improvements—such as user interface design and parameter automation—and the integration of more experimental technologies.

5.2 Final Considerations

While technologies such as cloud computing, advanced data engineering practices, and machine learning provide powerful means to reimagine organizational operations, their effective adoption requires aligning people, processes, and governance around a coherent vision of what it means to be truly data-driven. My six-month internship in Paris—situated in the centre of the luxury-brand ecosystem and embedded in a highly skilled squad within a large company—brought this lesson into sharp relief. The experience was professionally fulfilling: it enabled me to consolidate theoretical knowledge in production settings, to acquire practical proficiency with Google Cloud Platform and core data-engineering tools, and to observe how technical choices are conditioned by organizational constraints. Working alongside seasoned practitioners clarified pragmatic trade-offs in design, testing and deployment, and reinforced the importance of collaborative discipline and

craftsmanship in building resilient systems.

The cases examined here demonstrate that new technologies can accelerate operations and reduce costs; however, without clear provisions for interoperability and stewardship, these benefits risk being undermined by fragmentation or loss of trust. Thus, responsibility must extend beyond infrastructure design to the cultivation of resilient ecosystems in which data is applied with purpose.

More broadly, this work invites enterprises to consider their role as active shapers of the knowledge systems that inform strategic decision-making. Achieving this requires an ongoing commitment to align technological capability with organizational vision—an effort that is as much about governance and critical reflection as it is about engineering skill. Sustainable digital transformation will be defined not only by technological sophistication but by the institutional structures that sustain trust and cross-functional collaboration.

Finally, the research underscores the evolving relationship between automation and human expertise. As methods become more sophisticated, the objective should be to reallocate human effort toward higher-value strategic tasks rather than to eliminate it. By confronting current limitations in model flexibility and infrastructural efficiency, future systems can become more autonomous and more adaptable to the heterogeneous, dynamic nature of enterprise data. The findings presented here should therefore be read as a contribution to an ongoing continuum of innovation, in which data engineering and artificial intelligence converge to shape the future of organizations.

Bibliography

[1] *A Deep Dive Into Beauty*. 2023. URL: www.loreal.com/en/group/about-loreal (cit. on p. 5).

[2] *Commitments and Responsibilities: Fighting Climate Change*. 2023. URL: www.loreal.com/en/commitments-and-responsibilities/for-the-planet/fighting-climate-change (cit. on p. 6).

[3] *Commitments for Women in Science*. 2022. URL: www.loreal.com/en/news/commitments/for-women-in-science--25th-anniversary/ (cit. on p. 6).

[4] D. Laney, ed. *3D Data Management: Controlling Data Volume, Velocity and Variety*. Meta Group, 2001 (cit. on p. 10).

[5] Tilak Agerwala Manoj Muniswamaiah and Charles Tappert. «Big Data in Cloud Computing Review and Opportunities». In: *International Journal of Computer Science & Information Technology (IJCSIT) Vol 11, No 4*. Seidenberg School of CSIS, Pace University, White Plains, New York, August 2019 (cit. on p. 11).

[6] Jeffrey Dean and Sanjay Ghemawat. «MapReduce: Simplified Data Processing on Large Clusters». In: *OSDI'04: Sixth Symposium on Operating System Design and Implementation*. San Francisco, CA, 2004 (cit. on p. 12).

[7] Daniel Patrick. «The Evolutionary Journey of Big Data Solutions in the Cloud». In: *CloudComputing.media* (2023) (cit. on p. 16).

[8] Kevin G. «FBA Inventory Optimization Team Inventory management techniques and best practices». In: *Amazon* (2023) (cit. on p. 23).

[9] Hugo da Giao, Andre Flores, Rui Pereira, and Jacome Cunha, eds. *Chronicles of CI CD: A Deep Dive into its Usage Over Time*. Faculty of Engineering, University of Porto, Portugal, 2024 (cit. on p. 25).

[10] Bal Mukund Sharma, Krishnakumar KM, and Rashmi Panda. *Oracle Autonomous Database in Enterprise Architecture: Utilize Oracle Cloud Infrastructure Autonomous Databases for better consolidation, automation, and security*. Packt Publishing, 2022 (cit. on p. 26).

[11] Keith Pijanowski. *A Reference Architecture for AI/ML Data Infrastructure*. MinIO, 2025 (cit. on p. 28).

[12] Databricks. *Introduction to Data Lakes*. 2025 (cit. on p. 46).

List of Figures

List of Tables

Acknowledgements

I would like to express my sincere gratitude to Prof. Paolo Garza for having accepted to accompany me throughout the development of this thesis at Politecnico di Torino, offering his availability and invaluable support. My heartfelt thanks also go to my manager at L'Oréal, Richard de Morais, for guiding and supporting me during my internship, and for significantly contributing to my professional growth. Finally, I would like to dedicate a special thought to my family, who have always supported me with affection and encouraged me to give my best. A heartfelt thank you goes in particular to my mother who, with her wisdom, her advice, and her constant presence, has been an irreplaceable guide for me throughout every stage of this journey.

Ringraziamenti

Desidero esprimere la mia sincera gratitudine al Prof. Paolo Garza per aver accettato di accompagnarmi nel percorso di questa tesi al Politecnico di Torino, offrendo la sua disponibilità e il suo prezioso supporto. Un sentito ringraziamento va anche al mio manager di L'Oréal, Richard de Morais, per avermi guidato e affiancato durante lo sviluppo del tirocinio, contribuendo in modo significativo alla mia crescita professionale.
Infine voglio rivolgere un pensiero sincero alla mia famiglia, che mi ha sempre sostenuto con affetto e incoraggiato a dare il meglio di me. Un grazie dal profondo del cuore va in particolare a mia mamma, che con le sue perle di saggezza, i suoi consigli e la sua presenza costante è stata per me una guida insostituibile in ogni fase di questo cammino.