Autonomy is all you need

Romain Avouac, Frédéric Comte and Thomas Faria

Abstract Abstract here

1 Introduction

In recent years, the European Statistical System (ESS) has committed to leverage non-traditional data sources in order to improve the process of statistical production, an evolution that is encapsulated by the concept of Trusted Smart Statistics [31]. This dynamic is accompanied by innovations in the statistical processes, so as to be able to take advantage of the great potential of these new sources (greater timeliness, increased spatio-temporal resolution, etc.), but also to cope with their complexity or imperfections. At the forefront of these innovations are machine-learning methods and their promising uses in the coding and classification fields, data editing and imputation [14]. The multiple challenges faced by statistical institutes because of this evolution are addressed in the Bucharest Memorandum on Official Statistics in a Datafied Society (Trusted Smart Statistics), which predicts that "the variety of new data sources, computational paradigms and tools will require amendments to the statistical business architecture, processes, production models, IT infrastructures, methodological and quality frameworks, and the corresponding governance structures", and consequently invites the ESS to assess the required adaptations and prioritize them [8].

In line with these recommendations, much work has been done in the context of successive projects at the European level in order to operationalize the use of non-traditional data sources in the production of official statistics. Within the scope of the ESSnet Big Data II project (2018-2020), National Statistical Offices (NSOs) have

Romain Avouac
Insee, 88 avenue François Verdier, Montrouge, e-mail: romain.avouac@insee.fr

Thomas Faria
Insee, 88 avenue François Verdier, Montrouge, e-mail: thomas.faria@insee.fr

been working across a wide range of themes (online job vacancies, smart energy, tracking ships, etc.) in order to put together the building blocks for using these sources in actual production processes and identify their limitations [10]. However, while a substantial amount of work has been devoted to developing methodological frameworks [7, 33], quality guidelines [17] as well as devising business architectures that make third-party data acquisition more secure [30], not much has been said about the IT infrastructures and skills needed to properly deal with these new objects.

Big data sources, which are at the heart of Trusted Smart Statistics, have characteristics that, due to their volume, their velocity (speed of creation or renewal) or their variety (structured but also unstructured data, such as text and images), make them particularly complex to process. Besides, the "skills and competencies to automate, analyse, and optimize such complex systems are often not part of the traditional skill set of most National Statistical Offices" [3]. Not incidentally, an increasing number of public statisticians trained as data scientists have joined NSOs in recent years. Within its multiple meanings, the term “data scientist” reflects the increased involvement of statisticians in the IT development and orchestration of their data processing operations, beyond merely the design or validation phases [5]. However, the ability of these new data professionals to derive value from big data sources and/or machine learning methods is limited by several challenges.

A first challenge is related to the lack of proper IT infrastructures to tackle the new data sources that NSOs now have access to as well as the accompanying need for new statistical methods. For instance, big data sources require huge storage capacities and often rely on distributed computing frameworks to be processed, which generally cannot be provided by traditional IT infrastructures [22]. Similarly, the adoption of new statistical methods based on machine learning algorithms often require IT capacities (in particular, GPUs - graphical processing units) to massively parallelize computations [32].

Another major challenge is related to the difficulty of transitioning from innovative experiments to production-ready solutions. Even when statisticians have access to development environments in which they can readily experiment, the step towards putting the application or model in production is generally very large. Such examples highlight the need to make statisticians more autonomous regarding the orchestration of their processings as well as fostering a more direct collaboration between teams, as advocated by DevOps and DataOps approaches.

A third challenge is to foster reproducibility in official statistics production. This quality criterion involves devising processing solutions that can produce reproducible statistics on the one hand, and that can be shared with peers on the other hand.

- Final challenge : encourage and faciliate collaboration - Against that background, we argue that common theme : fostering autonomy - ref innovation plateformes blabla - choix technologiques qui favorisent l’autonomie et la scalabilité - make cloud resources easily available - retex : insee + ssp - MLOps case study to illustrate - open-source project - one-stop-shop - blueprint for building other similar data science platforms

- Thème général : donner de l’autonomie
- Limites du poste de travail : littérature sur scaling horizontal / vertical

- Observation commune aux différents INS :
  - Insee / SSM : homogénéité des parcours, pourtant grande diversité d'infra, de moyens DSI → difficulté à partager des environnements, des formations → idée de fournir une "sandbox", un commun technologique (2020) [NB : dans la continuité, sandbox à l'échelle européenne via le one-stop-shop (2024)]
  - Visions/incitations différentes DSI/statisticien → sécurité avant le fonctionnel
- Inspirations : DevOps, DataOps

2 Principles for building a modern and flexible data architecture for official statistics

With the emergence of big data sources and new methodologies offering significant promise to improve the production process of official statistics, statisticians trained in data science techniques are eager to innovate. However, their ability to do so is limited by several challenges. Central among these challenges is the need for greater autonomy — be it in scaling resources to match statistical workloads, deploying proofs of concept with agility and in a collaborative manner, etc. Against that background, our aim was to design a data science platform that not only manages big data efficiently but also empowers statisticians by enhancing their autonomy. To achieve this, we delved into the evolving data ecosystem in order to identify significant trends with the potential to overcome the aforementioned limitations[1]. Our findings indicate that leveraging cloud-native technologies, particularly containers and object storage, is key to building infrastructures capable of handling large and varied datasets in a flexible, cost-effective manner. Furthermore, these technologies significantly enhance autonomy, facilitating innovation and promoting reproducibility in the production of official statistics.

2.1 Limitations of traditional big data architectures

Over the last decade, the landscape of big data has dramatically transformed. Following the publication of Google's seminal papers that introduced the MapReduce paradigm [13, 6], Hadoop-based systems rapidly became the reference architecture of the big data ecosystem, celebrated for their capability to manage extensive datasets

[1] As a preamble to this review, we should note that, although we did our best to ground our insights in the academic literature, a lot of it stems from informal knowledge gathered through diligent and ongoing technology watch. In the rapidly evolving data ecosystem, traditional research papers are increasingly giving way to blog posts as the primary references for cutting-edge developments. This shift is largely due to the swift pace at which big data technologies and methodologies are advancing, making the lengthy publication process of formal research often not the preferred way of disseminating timely insights and innovations.

through the use of distributed computing. The inception of Hadoop marked a revolutionary step, enabling organizations to process and analyze data at an unprecedented scale. Basically, Hadoop provided companies with all-rounded capabilities for big data analytics: tools for ingestion, data storage (HDFS), and computing capacities (Spark, among others) [9], thus explaining its rapid adoption across industries.

In the late 2010's, Hadoop-based architectures have experienced a clear decline in popularity. In traditional Hadoop environments, storage and compute were co-localized by design: if the source file is distributed across multiple servers (horizontal scaling), each section of the source file is directly processed on the machine hosting that section, so as to avoid network transitions between servers. In this paradigm, scaling the architecture often meant a linear increase in both compute and storage, regardless of the actual demand. In a recent article provocatively titled "Big Data is Dead"[2], Jordan Tigani, one of the founding engineers behind Google BigQuery, explains why this model doesn't fit the reality of most data-centric organizations anymore. First, because "in practice data sizes increase much faster than compute sizes". While the amount of data generated and thus needing to be stored may grow linearly over time, it is generally the case that we only need to query the most recent portions of it, or only some columns and/or groups of rows. Besides, Tigani points out that "the big data frontier keeps receding": advancements in server computing capabilities and declining hardware costs mean that the number of workloads that don't fit on a single machine — a simple yet effective definition of big data — has been continually decreasing. As a result, by properly separating storage and compute functions, even substantial data processing jobs may end up using "far less compute than anticipated [...] and might not even need to use distributed processing at all".

These insights strongly align with our own observations at Insee in recent years. As a use case of using big data infrastructures to improve statistical processes, an Insee team set up a Hadoop cluster as an alternative architecture to the one already in use to process sales receipt data in the context of computing the consumer price index. An acceleration of data processing operations by up to a factor of 10 was achieved, for operations that previously took several hours to perform [19]. Despite this increase in performance, this type of architectures were not reused later for several reasons. Mainly, the architecture proved to be expensive and complex to maintain, necessitating specialized technical expertise rarely found within NSOs [35]. But interestingly, subsequent projects involving large datasets didn't suffer much from this change, as their needs were actually very much in line with Tigani's observations. The bottleneck for these projects was generally on the side of computational needs rather than storage capacity. Furthermore, although these projects could still involve substantial data volumes, we observed that effective processing could be achieved using conventional software tools (R, Python) on single-node systems by leveraging recent promising tools from the data ecosystem. First, by using efficient formats to store the data such as Apache Parquet [11], which properties (columnar storage [1], optimisation for the "write once, read many" (WORM) paradigm, ability to partition data, etc.) make it particularly suited to analytical tasks such as those

[2] https://motherduck.com/blog/big-data-is-dead/

generally performed in official statistics [2]. Second, by performing computations using in-memory computation frameworks such as Apache Arrow [12] or DuckDB [29], that are also based on columnar representation — thus working in synergy with Parquet files — and implementing various optimizations (predicate pushdown, projections pushdown) to limit computations to data effectively needed, enabling much larger-than-memory data processing on usual, single-node machines.

2.2 Embracing cloud-native technologies

In light of this evolution of the big data ecosystem, there has been a notable shift in recent years within the industry towards more flexible and loosely coupled architectures. The advent of cloud technologies has been instrumental in facilitating this shift. Unlike the era where Hadoop was prominent, network latency has become much less of a concern, making the traditional model of on-premise and co-located storage and compute solutions less relevant. In terms of the nature of the data that need to be processed, we are observing an evolution that some have described as moving "from big data to flexible data": modern data infrastructures are required not only to process large volumes but also to be adaptable in multiple dimensions: accommodating various data structures (ranging from structured, tabular formats to unstructured formats like text and images), ensuring data portability across multi-cloud and hybrid cloud environments, and supporting a diverse range of computational workloads (from parallel computations to deep learning models necessitating GPUs, as well as the deployment and management of applications) [21]. In recent years, two technologies have emerged in the data ecosystem as foundational technologies for achieving such flexibility in cloud-based environments: containerization and object storage.

In a cloud environment, the computer of the user becomes a simple access point to perform computations on a central infrastructure. This enables both ubiquitous access to and scalability of the services, as it is easier to scale a central infrastructure — usually horizontally, i.e. by adding more servers. However, such centralized infrastructures have two well-identified limitations that need to be dealt with: the competition between users in access to physical resources and the need to properly isolate deployed applications. The choice of containerization is fundamental as it tackles these two issues [4]. Fundamentally, a container is a logical grouping of resources that makes it possible to encapsulate an application, its libraries and other system dependencies, in a single package. By creating “bubbles” specific to each service, containers thus guarantee application isolation while remaining lightweight, as they share the support operating system with the host machine — contrary to virtual machines (see. graph X). In order to manage multiple containerized applications in a systematic way, containerized infrastructures generally rely on an orchestrator software — the most prominent one being Kubernetes, an open-source project initially developed by Google to manage its numerous containerized workloads in production [36]. Orchestrators automate the process of deploying, scaling, and managing containerized applications, coordinating their execution across various servers. Interestingly, this property makes it possible to handle very large volumes of data in a distributed way: containers break down big data processing operations into a multitude of small tasks, organized by the orchestrator, thus minimizing the required resources while providing more flexibility than hadoop-based architectures [37].

The other fundamental choice in a data architecture is the nature of data storage. In the cloud ecosystem, so-called "object storage" has become the de-facto reference [34] [3]. In this paradigm, files are stored as "objects" consisting of data, an identifier and metadata. This type of storage is optimized for scalability, as objects are not limited in size and the underlying technology enables cost-effective storage of (potentially very) large files. It is also instrumental in building a decoupled infrastructure such as discussed before: the data repositories — referred to as "buckets" — are directly searchable using standard HTTP requests through a standardized REST API. In a world where network latency is not the main bottleneck anymore, this means that storage and compute don't have to be on the same machines or even in the same location, and can thus scale independently according to specific organization demands. Finally, object storage is a natural complement to architectures based on containerised environments for which it provides a persistence layer — containers being stateless by design — and easy connectivity without compromising security, or even with strengthened security compared with a traditional storage system [25].

2.3 Leveraging cloud technologies to increase autonomy and foster reproducibility

Understanding how the technological choices described in the technical discussion above are relevant in the context of official statistics require an in-depth review of statisticians' professional practices in their use of computing environments. At the end of the 2000s, with micro-computing at its peak, many of the technical resources used by statisticians at Insee were local: the code and processing software were located on individual computers, while data was accessed through a file-sharing system. Because of the the limited scalability of personal computers, this setup greatly limited the ability of statisticians to experiment with big data sources or computationally intensive statistical methods, and involved security risks because of the widespread data dissemination within the organization. In order to overcome these limitations, a transition was made towards centralised IT infrastructures, concentrating all — and thus overall much more — resources on central servers. Such infrastructures, made available to statisticians through a shared, virtual desktop environment for ease of use, remains the dominant method for conducting statistical computations at Insee at the time of writing this article.

Through our observations and discussions with fellow statisticians, it became evident that although the current IT infrastructure adequately supported the core activities of statistical production, it noticeably restricted statisticians' capacity to

[3] Mainly because of Amazon's "S3" (Simple Storage Service) implementation.

experiment freely and innovate. The primary bottleneck in this organization is the dependency of statistical projects on centralized IT decision-making, such as the allocation of computing resources, access to shared data storage, the use of pre-configured programming languages and packaging environments, etc. Besides, such dependencies often lead to a well-known phenomenon within the software development community that lies at the heart of the DevOps approach, where the priorities of developers — iterate rapidly to improve functionality in a continuous manner — often clash with IT's focus on security and process stability. On the contrary, it is our understanding that modern data science practices reflect an increased involvement of statisticians in the IT development and orchestration of their data processing operations, beyond merely the design or validation phases. New data science infrastructures must take this expanded role of their users into account, giving them more autonomy than conventional infrastructures.

Cloud technologies stand out as a powerful solution to grant statisticians this much-needed autonomy in their daily work, enabling a culture of innovation. Through object storage, users gain control over the storage layer, allowing them to experiment with diverse datasets without being constrained by the limited storage spaces typically allocated by IT departments. Containerization empowers users to customize their working environments to their specific needs — be it programming languages, system libraries, or package versions — while also providing the flexibility to scale their applications according to the required computing power and storage capacities. By design, containers also foster the development of portable applications, which enables smoother transitions between environments (development, qualification, production), ensuring that applications can be moved seamlessly without the hurdles of environmental inconsistencies. Finally, with orchestration tools like Kubernetes, statisticians can more readily deploy applications and APIs and automatize the whole building process, sidestepping complexities associated with inconsistent or complex deployment environments. This capability aligns with the DevOps approach, enabling quicker iteration and building minimal prototypes as proofs of concept (POCs) rather than building the optimal (but time-consuming) solution for a pre-defined objective [20].

Besides scalability and autonomy, these architectural choices also foster reproducibility of statistical computations. The concept of reproducibility — namely the ability to reproduce the result of an experiment by applying the same methodology to the same data — is a fundamental criterion of scientific validity [24]. It is also highly relevant in official statistics, as it serves as a foundation for transparency, which in turn is crucial for building and maintaining the public's trust. Fostering reproductibility in statistical production involves devising processing solutions that can produce reproducible statistics on the one hand, and that can be shared with peers on the other hand [23]. Traditional IT infrastructures — either a personal computer or a shared infrastructure with remote desktop access — fall short in this regard, as building a project or just computing a statistical indicator there generally involves a series of manual steps (installing system libraries, the programming language binary, projects packages, dealing with potentially conflicting versions, etc.) that can not be fully reproduced across projects. In comparison, containers are reproducible by design, as their build process involves defining precisely all the needed resources as a set of processing operations in a standardized manner, from the "bare machine" to the running application [26]. Furthermore, these reproducible environments can be easily shared to peers as they can be readily published on open registries (for example, a container registry such as DockerHub) along to the source code of the application (for example, on a public software forge like GitHub or GitLab). This approach significantly enhances the reusability of code projects, fostering a community-driven model of development and innovation.

3 Onyxia: an open source project to build cloud-native data science platforms

The Onyxia project, initiated by the French Public Service and available at onyxia.sh, is an open source project aimed at creating self-sufficient data science environments in the cloud or on-premises. This project can be seen as a “Platform as a Package” (PaaP) solution for organisations wishing to create a data science environment based on cloud technologies.

3.1 Making cloud-technologies accessible to statisticians

Our technology watch and literature review highlighted cloud-native technologies, in particular containerization and object storage, as instrumental in building a data science platform that is both scalable and flexible. Building on these insights, we established our initial on-premise Kubernetes cluster in 2020, integrating it with MinIO, an open-source object storage system designed to work seamlessly with Kubernetes. Yet, our first experiments highlighted a significant barrier to their widespread adoption: the complexity of their integration. This is an important consideration when building data architectures that prioritize modularity — an essential feature for the flexibility we aim to achieve. For instance, due to MinIO’s compatibility with the Amazon S3 API, the storage source could easily be switched without requiring substantial modifications to one managed by a public cloud provider. However, modularity of the architecture components also entails that any data application launched on the cluster must be configured so as to communicate with all the components. For instance, in a big data setup, configuring Spark to operate on Kubernetes while interacting with datasets stored in MinIO requires an intricate set of configurations (specifying endpoints, access tokens, etc.), a skill set that typically lies beyond the expertise of statisticians.

This very insight is really the base of the Onyxia project : choosing technologies that foster autonomy won’t actually foster autonomy if their complexity acts as a deterrent from widespread adoption in the structure. In recent years, statisticians at Insee already need to adapt to a changing environment in termes of their everyday tools : transitioning from proprietary software (SAS) to open-source ones (R, Python), acculturating to technologies that improve reproducibility (version control with Git), consuming and developing APIs, etc. These changes, that make their activity more and more akin to the one of software developers, already imply significant training and changes in the modalities of work everyday. In this regard, adoption of cloud-technologies was utterly dependent on making them readily accessible.

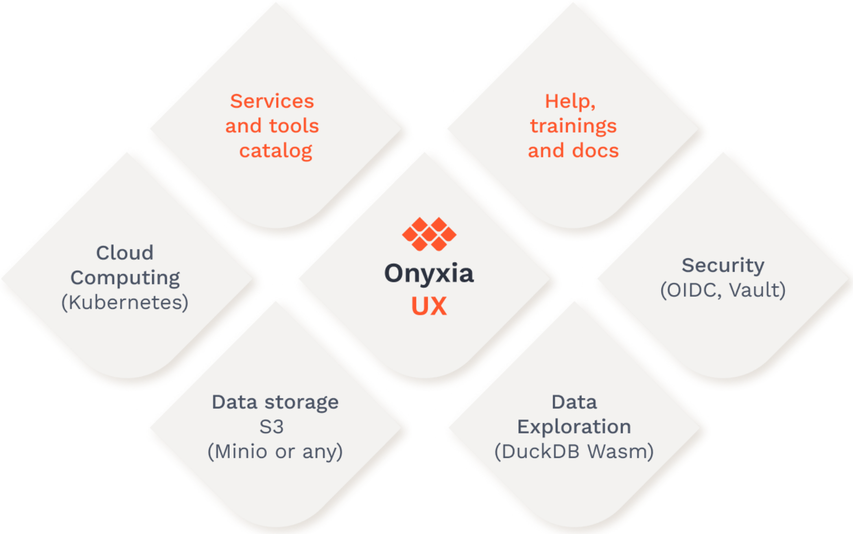
Services and tools catalog
Help, trainings and docs
Cloud Computing (Kubernetes)
Onyxia UX
Security (OIDC, Vault)
Data storage S3 (Minio or any)
Data Exploration (DuckDB Wasm)

Fig. 1 Onyxia is the technical binder between cloud-native modular components

To bridge this gap, we developed Onyxia, an application that essentially acts as interface between the modular components that compose the architecture (see fig 1). The main entrypoint of the user is a user-friendly web application[4] that enables users to launch services from a data science catalog (see section 3.3) as running containers on the underlying Kubernetes cluster. The interface between the UI and Kubernetes is done by a lightweight custom API[5], that essentially transforms the application request of the user into a set of manifests to deploy Kubernetes resources. For a given application, these resources are packaged under the form of Helm charts, a popular way of packaging potentially complex applications on Kubernetes [15]. Although users can configure a service to tailor it to their needs, they will most of the time just launch a service out-of-the-box and be able to start developing. This point really illustrates the added value of Onyxia in facilitating the adoption of cloud technologies. By injecting authentication information and configuration into the containers at the initialization, we ensure that users can launch and manage

[4] `https://github.com/InseeFrLab/onyxia-ui`

[5] `https://github.com/InseeFrLab/onyxia-api`

data science services in which they can interact seamlessly with the data from their bucket on MinIO, their sensitive information (tokens, passwords) stored in Vault, etc. This automatic injection, coupled with the pre-configuration of data science environments in Onyxia's catalogs of images[6] and associated helm-charts[7], make it possible for users to execute potentially complex workloads - such as running distributed computations with Spark on Kubernetes using data stored in S3, or training deep-learning models using a GPU - without getting bogged down by the technicalities of configuration.

3.2 Architectural choices aimed at fostering autonomy

The Onyxia project is based on a few structuring principles, with a central theme : fostering autonomy. First, at the level of the organization by preventing vendor lock-in. In order to get a competitive edge, many commercial cloud providers develop applications and protocols that customers need to use to access cloud resources but that are not interoperable, greatly complexifying potential migrations to another cloud platform [28]. Recognizing these challenges, there is a trend towards endorsing cloud-neutral strategies [27] in order to reduce reliance on a single vendor's specific solutions. In contrast, the use of Onyxia is inherently not restrictive: when an organization chooses to use it, it chooses the underlying technologies - containerization and object storage - but not the solution. The platform can be deployed on any Kubernetes cluster, either on-premise or in public clouds. Similarly, although Onyxia was designed to be used with MinIO because it is an open-source object-storage solution, but is also compatible with objects storage solutions from various cloud providers (AWS, GCP).

The other important level at which Onyxia fosters autonomy is at the level of users. Proprietary softwares that have been used intensively in official statistics - such as SAS or STATA - also produce a vendor lock-in phenomenon. The costs of licensing are high and can evolve quickly, and users are tied in certain ways of performing computations, preventing progressive upskilling. On the contrary, Onyxia aspires to be removable; we want to enhance users' familiarity and comfort with the underlying cloud technologies rather than act as a permanent fixture in their workflow. An illustrative example of this philosophy is the platform's approach to user actions: for tasks performed through the UI, such as launching a service or managing data, we provide users with the equivalent terminal commands, promoting a deeper understanding of what actually happens on the infrastructure when triggering something. Furthermore, all the services offered through Onyxia's catalog are open-source.

Naturally, the way Onyxia makes statisticians more autonomous in their work depends on their needs and familiarity with IT skills. Statisticians that just want to have access to extensive computational resources to experiment with new data

[6] https://github.com/InseeFrLab/images-datascience

[7] https://github.com/InseeFrLab/helm-charts-interactive-services

sources or statistical methods will have access in a few clicks to easy-to-use, preconfigured data science environments, so that they can directly start to work and prototype their solution. However, many users want to go deeper and build actual prototypes of production applications for their projects: configuring initialization scripts to tailor the environments to their needs, deploying an interactive app that delivers data visualisation to users of their choice, deploying other services than those available in our catalogs, etc. For these advanced users to continue to push the boundaries of innovation, Onyxia gives them access to the underlying Kubernetes cluster. This means that users can freely open a terminal on an interactive service and interacts with the cluster - within the boundaries of their namespace - in order to apply custom resources and deploy custom applications or services.

Besides autonomy and scalability, the architectural choices of Onyxia also foster reproducibility of statistical computations. In the paradigm of containers, the user must learn to deal with resources which are by nature ephemeral, since they only exist at the time of their actual mobilization. This fosters the adoption of development best practices, notably the separation of the code — put on an internal or open-source forge such as GitLab or GitHub — the data — persisted on a specific storage solution, such as MinIO — and the computing environment. While this requires an entry cost for users, it also helps them to conceive their projects as pipelines, i.e. a series of sequential steps with well-defined inputs and outputs (akin to directed acyclic graph (DAG)). The projects developed in that manner are usually more reproducible and portable — they can work seamlessly on different computing environments — and thus also more readily shareable with peers.

3.3 An extensive catalogue of services to cover the entire lifecycle of data science projects

In developing the Onyxia platform, our intention was to provide statisticians with a comprehensive environment designed to support the prototyping of their data science projects from start to finish. As depicted in Figure 2, the platform offers a vast array of services that span the complete lifecycle of a data science project.

The primary usage of the platform is the deployment of interactive development environments (IDE), such as RStudio, Jupyter, or VSCode. These IDEs come equipped with the latest kernels of major open-source programming languages commonly employed by public statisticians (R, Python, Julia), as well as an extensive collection of packages commonly used in data science for each language. In order to ensure that services remain up-to-date and consistent between them, we maintain our own stack of underlying Docker images and rebuilt it weekly. The stack of image is fully open-source and can thus be reused outside of Onyxia[8].

As discussed in previous sections, the persistence layer of these interactive environments is mainly carried out by MinIO, Onyxia's default object storage solution.

[8] `https://github.com/InseeFrLab/images-datascience`

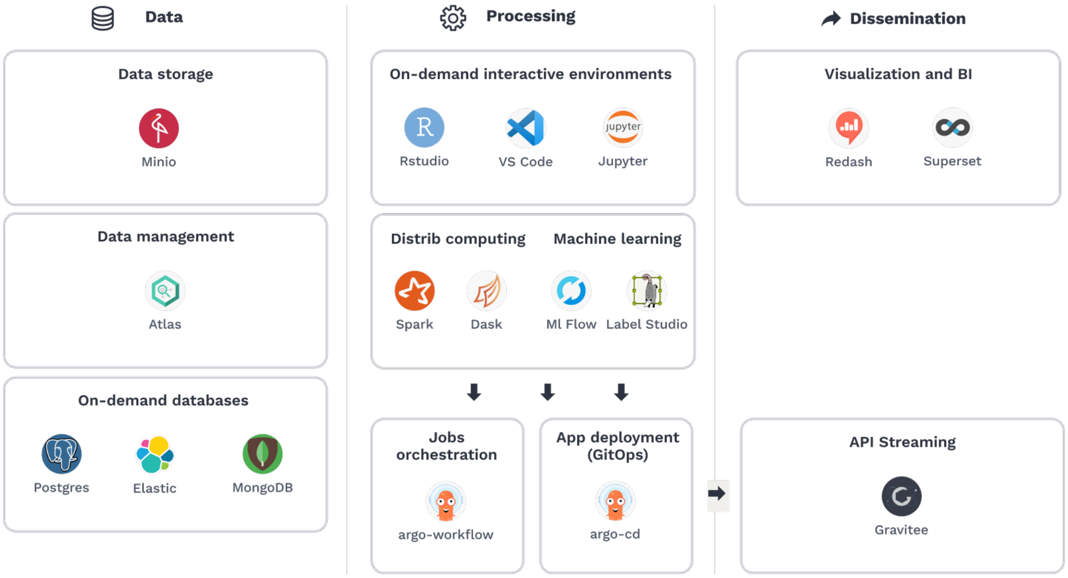

Fig. 2 Onyxia's catalog aims at covering the entire lifecycle of data science projects

As it is based on a standardized REST API, files can be easily queried directly from R or Python using high-level packages. This in itself is an important step of ensuring reproducibility: the input files of a project are not mounted manually and then specified via paths adherent to a specific infrastructure and filesystem. Rather, files are specified as HTTP queries, making the overall structure of projects much more extendable. In our experience, the object-storage paradigm covers very well the needs of most statistical projects we accompany. However, additional database services such as PostgreSQL and MongoDB are available for applications with specific needs, such as those requiring online transaction processing (OLTP) capabilities or document-oriented storage.

As Onyxia was developed to allow experimentation with big data sources and machine learning methods, we also provide services optimized for scalability. For instance, frameworks like Spark and Trino that enable to perform distributed computations within Kubernetes. These services come pre-configured to integrate seamlessly with S3 storage, thus facilitating building integrated and efficient data pipelines.

Beyond mere experimentation, our goal is to empower statisticians to transition from trial phases to production-ready projects. In lines with principles from the DevOps approach, this involves facilitating the deployment of prototypes and their continuous improvement over time. To this end, we provide a set of open-source tools aimed at automatizing and industrializing the process of deploying data-intensive applications (ArgoCD, Argo-Workflows, MLFlow). For projects leveraging machine-learning models, statisticians can serve their models through APIs, deploy them using the aforementioned tools, and and manage their lifecycle using an API manager such as Gravitee. Section 4 will delve into how these tools, particularly MLFlow, have been instrumental in putting machine learning models in production at Insee, in accordance with MLOps principles.

In section 3.2, we stressed that one of Onyxia's fundamental design principle was to avoid vendor lock-in. In line with this idea, organizations that implement Onyxia are at liberty to customize catalogs to suit their specific requirements, or even opt to construct their own catalogs independent of Onyxia's default offerings. This flexibility ensures that organizations are not confined to a single solution or provider, and can adapt the platform to their evolving needs.

3.4 Building commons : an open-source project and an open-innovation platform

In many ways, the Onyxia project aims at building commons in the realms of official statistics. First of all, by being a fully open-source initiative [].

- Utilisation de communs : briques open-source - Projet open-source -> permettre des réinstanciations dans d'autres orgas

- SSP CLoud - Orientation plateforme : instance vivante d'Onyxia, ouverte, collaborative, sandbox (cf. ref papier SSP Cloud sur l'aspect plateforme) - Innovation ouverte → littérature - Open-data - Instance de partage : formations reproductibles + utilisation dans les écoles de stats + hackathons (organisation annuelle du funathon cf. one-stop-shop)

4 Case-study : deploying a machine learning model into production following MLOps principles

This chapter aims, through a concrete example, to illustrate how INSEE managed to deploy its first machine learning model into production. It will delve into the MLOps approach that this project strived to adhere to as much as possible, focusing on the various technologies and infrastructures that were employed. This initial production deployment, while successful, faced various challenges, whether technical or organizational, and we will endeavor to discuss them and propose solutions wherever possible. The idea is to illustrate the development of this project as transparently as possible, without claiming it to be the definitive approach. The entire project is available in open source[9] and remains under active development.

[9] `https://github.com/orgs/InseeFrLab/teams/codification-ape/repositories`

4.1 Context and motivations

Coding tasks are common operations for all national statistical institutes and can sometimes be challenging due to the size of certain nomenclature. At INSEE, a highly sophisticated coding tool called Sicore was developed in the 1990s to perform various classifications. Sicore uses a reference file that can be considered as a training file, which serves as examples of codings. The label to be coded is compared to the labels contained in the training file, and when the label is recognized, the associated code is assigned. When the label is not recognized, it must be manually classified by an INSEE agent. Two main reasons drove the experimentation of new coding methods. Firstly, there was an internal change with the redesign of the Sirene registry, which lists all companies in France and assigns them a unique identifier, the Siren number, for use by public institutions, notably to improve the daily management of the registry for INSEE agents and to reduce waiting times for companies. Additionally, at the national level, the government launched a one-stop shop for business formalities, allowing more flexibility for business owners in describing their main activities.

The initial testing exercises revealed that Sicore was no longer the suitable tool for performing NACE classification, as only 30% of tasks were being automatically coded. The teams working on the Sirene registry were already overwhelmed with numerous changes, making it unrealistic to further increase their workload with manual reclassification, which is both time-consuming and unstimulating. Therefore, in May 2022, the decision was made to experiment with new methods for performing this classification task, with the aim of using this method in production by January 1, 2023, the launch date of the new Sirene registry, if successful.

This choice of innovation was not initially a voluntary decision but rather a necessity, given that the current state of the process could not remain unchanged. Therefore, all decisions made during this project were taken considering these temporal and organizational constraints. The aim is to present these various strategic choices that we made at INSEE while bearing in mind that they may not be applicable or advisable in all organizations.

Three stakeholders were involved in this project: the business team responsible for managing the Sirene registry, the IT team developing software related to the registry's operation, and the *"innovation"* team tasked with implementing the new coding tool. The latter team is the INSEE Lab, which was created in 2017 with the objective of providing support to other teams on innovation topics to streamline their various projects.

4.2 Démarrage du projet comme les projets expérimental et prise en compte des contraintes

The project we aim to implement is a standard natural language classification problem. Indeed, starting from a textual description, we want to predict the class associated with it in the NACE Rev. 2 nomenclature. This nomenclature has the particularity of being hierarchical and containing 5 different levels[10]: section, division, group, class, and subclass. In total, 732 subclasses exist, which is the level at which we aim to perform our classification. Table 1 summarizes this hierarchical structure with an example.

Level	**NACE**	**Title**	**Size**
Section	H	Transportation and storage	21
Division	52	Warehousing and support activities for transportation	88
Group	522	Support activities for transportation	272
Class	5224	Cargo handling	615
Subclass	**5224A**	**Harbour handling**	**732**

Table 1 NACE Nomenclature

With the establishment of the one-stop shop, business owners can now freely draft their activity descriptions. As a result, the labels received by INSEE are very different from the harmonized labels that were previously received. Therefore, it was decided to work with machine learning models that have proven their effectiveness in the literature. This represents a significant paradigm shift from INSEE's perspective, as no machine learning model has ever been deployed into production. In fact, the initial years of the INSEE Lab were characterized by a multitude of experiments on various subjects without ever transitioning to production. Nevertheless, all these experiments were valuable and accelerated the project through the gained experience. This project thus marked the first instance where the challenges of production deployment were considered from the outset, guiding numerous methodological and technical choices. As such, several points had to be agreed upon to facilitate coordination among the various stakeholders, and several strategic choices had to be made from the outset, including the working infrastructure, work methods, and the type of model to be used for such a project. The goal was to accommodate the constraints of each team and reconcile their needs.

4.2.1 Working infrastructure

In a machine learning project, the choice of development infrastructure is central. Working on a modular infrastructure is crucial in machine learning projects due to the diversity of tasks to be performed, such as data collection, preprocessing, modeling, evaluation, inference, monitoring, among others. This allows for easy replacement or updating of components without disrupting the entire workflow pipeline. As elucidated in the preceding chapter, traditional Big Data infrastructures often prove too rigid and specialized, failing to adequately cater to the diverse demands of various projects. Therefore, we decided to use a more modular infrastructure that better addresses the needs of machine learning projects by leveraging the most appropriate technologies for each stage of our pipeline. This primarily allows us

[10] Actually, there are 5 different levels in France but only 4 at the European level.

to take advantage of the latest technological advancements without being limited by a predefined architecture, as innovations in machine learning progress rapidly. We exclusively utilized open-source cloud technologies available in the SSP Cloud catalog, an instance of the Onyxia software developed by INSEE. This platform, based on Kubernetes, offers flexible and scalable container management, enabling easy scaling of services as needed. As we will see in the following sections of this chapter, we used various tools for each stage of our pipeline: MinIO, Vault, MLflow, ArgoCD, Argo Workflows, Label Studio, Vscode, etc. The decision to opt for the SSP Cloud was also motivated by the ongoing implementation of a private instance of Onyxia on the production servers at INSEE. This initiative is poised to streamline the connection between production and development environments in the long run.

While the utilization of SSP Cloud has proven to be the right solution for us, it does come with its set of considerations. Working on SSP Cloud implies operating in ephemeral environments, thus ensuring the proper backup of code and data is imperative. Thankfully, SSP Cloud provides a file storage solution through the use of MinIO, which addresses this requirement. Although Onyxia simplifies initial setup and usage, it may require a bit of a learning curve for new users. However, given the widespread adoption of Amazon's S3 storage system in cloud computing, investing time in understanding it proves beneficial for any data scientist. Furthermore, comprehensive documentation is available at the following address: `https://inseefrlab.github.io/docs.sspcloud.fr/docs/en/storage.html`. Similarly, regular versioning of code is essential. We've opted for Git and Github for source code management, enabling seamless collaboration among our teams. While Git may require some initial learning, it's a crucial tool for anyone working on data science projects. To facilitate its adoption within INSEE, the innovation teams developed an internal training session on Git usage. For specific documentation related to SSP Cloud, it can be found here.

As for data privacy, while SSP Cloud is secure, it doesn't guarantee complete confidentiality. However, since we solely deal with open data, this hasn't posed any issues for us. At the time, the internal Onyxia instance at INSEE was not available yet.

4.2.2 Working methods and good practices

As mentioned earlier, our working infrastructure essentially required us to use Git to version our code. However, regardless of your setup, knowledge and usage of Git are essential prerequisites for any machine learning project. Another significant choice was made at the project's outset: the programming language to use. Currently, INSEE is undertaking a large-scale project to migrate all SAS code to the open-source R language. While we could have followed suit to align our projects with other INSEE initiatives, we opted for Python. Without getting into the futile debate over the superiority between R and Python, it's generally accepted that the majority of the machine learning ecosystem leans towards Python. That's why we chose this language. However, this decision wasn't made lightly, as it means our three stakeholders primarily work with three different programming languages: Java for IT teams, R for business teams, and Python for innovation teams. This diversity can pose a significant challenge to collaboration among the three teams, and we'll explain how we've attempted to overcome this hurdle.

In this project, we deliberately favored the use of scripts over notebooks, even though we're accustomed to using the latter in other projects. With a focus on production deployment, where our main goal was to ensure code scalability and efficient maintenance, notebooks present several significant limitations, including:

1. Defining and making all objects (functions, classes, and data) available in the same file, thereby complicating long-term code maintenance.
2. Limited potential for automating ML pipelines.
3. Notebooks' tendency to encourage code duplication and marginal modifications rather than using functions, making the code less modular and harder to maintain.
4. Lack of extensions to implement best practices, such as linters, making it challenging to apply code quality standards.
5. The costly transition to production with notebooks, whereas well-structured scripts are easier to deploy.
6. Major versioning challenges with Git, as notebooks are essentially large JSON files, making it difficult to identify code changes and thus collaborate among team members.

Moreover, by open-sourcing all our code, we've committed to following community standards by documenting our code and using formatters such as Ruff.

4.2.3 Methodology used

When it came to selecting the methodology to adopt, we had to navigate through various constraints, with the most significant being the need for deployment on our production servers. Our model had to be lightweight enough to run efficiently on these servers, while also minimizing the computational resource overhead to ensure swift inference. Additionally, the model had to be compatible with a different language than the one it was trained, namely Java, utilized on our production servers. This proved particularly challenging throughout our project, as it influenced our data preprocessing choices, necessitating simplicity wherever possible. We had to meticulously replicate the data processing performed in Python during training, thereby limiting the use of certain packages. Ultimately, each decision made had to ensure the model's compatibility and efficiency within a production environment while minimizing compromises on inference quality.

These constraints led us away from the most powerful language models at the start of the project, such as Transformer models, and instead directed us towards simpler natural language models, specifically, we opted for the fastText model [16]. This choice was driven by its ability to address all previously mentioned constraints. The fastText model is incredibly fast to train, even from scratch, and inference doesn't require a GPU to be extremely rapid. Moreover, there exists a wrapper that enables reading fastText models in Java, which could be utilized by the IT teams on their machines, greatly facilitating the deployment of our models. Unfortunately, this wrapper, available on GitHub, is no longer maintained, posing serious security concerns, and thus became a temporary default choice for us. In addition to these technical arguments, the decision to use the fastText model was justified from a methodological standpoint. Firstly, the INSEE Lab teams had already completed several projects using the fastText model, leveraging the acquired knowledge to achieve initial results very quickly. While it may not have been a state-of-the-art language model, for our use case, the model yielded excellent performance results which, considering the time and human resource constraints, were more than sufficient to enhance the existing process. Finally, the model is inherently simple methodologically speaking, greatly simplifying communication and adoption within the various INSEE teams.

The supervised classification model fastText relies on both a bag-of-words model to obtain embeddings and a classifier based on logistic regression. The bag-of-words approach involves representing a text as the set of vector representations of each of its constituent words. Thus, the embedding of a sentence depends on the embeddings of its words, for example, their sum or their average. In the case of supervised text classification, the embedding matrix and the classifier's coefficient matrix are learned simultaneously during training by gradient descent, minimizing an usual cross-entropy loss function. The specificity of the fastText model lies in embeddings being performed not only on words but also on word n-grams and character n-grams, providing more context and reducing biases due to spelling mistakes. The fastText model can be summarized by the following diagram:

4.3 What is MLOps?

4.3.1 Specificities of ML projects

In the landscape of machine learning projects, there are new considerations and challenges compared to traditional statistical projects. These new challenges stem from the complex nature of ML models, their iterative development process, and the need for automation and scalability. Among others, here are some key aspects in ML projects:

1. **Logging Parameters**: ML models often have numerous hyperparameters and configurations that significantly impact their performance. It's crucial to log these parameters along with model training and evaluation metrics to ensure reproducibility and traceability. This logging allows for better understanding of model behavior and facilitates debugging and optimization efforts.
2. **Optimizing Hyperparameters**: Tuning hyperparameters is a critical step in optimizing the performance of ML models. Unlike traditional statistical models where parameters are often predefined, ML models typically have a larger number of hyperparameters that need to be optimized. This process requires sophisticated

techniques such as grid search, random search, or Bayesian optimization, which can be computationally intensive and time-consuming.

3. **Model Versioning**: ML models are highly iterative. Therefore, effective version control mechanisms are essential to track changes, compare performance across different model versions, and roll back to previous versions if necessary. While model versioning may also be relevant for traditional statistical models, it is infrequently practiced. In ML projects, however, it is considered mandatory due to the iterative nature of model development and the dynamic nature of data and algorithms.
4. **Model Deployment**: Deploying ML models into production environments presents unique challenges due to their complexity and resource requirements. Ensuring seamless deployment involves considerations such as containerization, scalability, latency, and integration with existing systems.
5. **Model Monitoring**: Once deployed, ML models need to be continuously monitored to detect performance degradation, concept drift, or other anomalies. Unlike traditional statistical models, ML models can exhibit unexpected behavior over time due to changes in data distribution or underlying patterns. Implementing robust monitoring solutions involves tracking model performance metrics and data quality to ensure ongoing reliability, effectiveness and to trigger model retraining.
6. **Model Retraining**: Regular retraining of ML models is essential to maintain their effectiveness over time. Changes in data distribution, data quality, or business objectives may necessitate updates to the model. Retraining ensures that the model remains accurate and relevant in evolving environments.
7. **Data annotation**:
8. **Explainability**: The interpretability of ML models is important for building trust and understanding their predictions.

4.3.2 From DevOps to MLOps

The MLOps approach is built upon the foundations of the DevOps approach. In this sense, it can be considered simply as an extension of DevOps, developed to address the specific challenges related to managing the lifecycle of machine learning models. MLOps incorporates the principles of collaboration and automation inherent in DevOps but also considers all aspects related to data and machine learning models.

MLOps involves the automation of tasks such as data management, tracking model versions, their deployments, as well as the continuous evaluation of model performance in production. Similar to DevOps, MLOps emphasizes close collaboration between business units and IT teams on one hand, and data science teams on the other. This collaboration is key to ensuring effective communication throughout the lifecycle of the machine learning model.

4.3.3 MLOps Principles

The pillars of MLOps[18] are generally defined as :

- **Reproducibility**: The results of every experiment, successful or unsuccessful, should be reproducible at no cost. This implies first and foremost a certain rigor in managing packages, environments, system libraries, code version control, etc. Additionally, the various variables influencing model training (training data, hyperparameters, etc.) must be versioned with the model.
- **Automation**: To foster feedback loops of continuous improvement, the model lifecycle (testing, building, validation, deployment) must be automated to the fullest extent possible. Tools derived from the DevOps approach, especially continuous integration and continuous deployment (CI/CD), should be leveraged.
- **Collaboration**: MLOps values a culture of collaborative work around ML projects, where communication within teams should reduce siloed work. On a technical level, MLOps tools used should facilitate collaborative work on data, model, and code used by the project.
- **Continuous Improvement**: Once deployed, it is essential to ensure that the model performs as expected by evaluating its performance on real data using continuous monitoring tools. In case of performance degradation over time, periodic retraining or continuous training of the model should be considered.

Tackling these challenges and effectively managing the lifecycle of a ML project necessitates specialized tools, many of which are accessible as open-source solutions. Throughout our project, we aimed to embrace the MLOps approach by utilizing software applications grounded in cloud-native technologies and available in the SSP Cloud catalog. This enables data scientists to be autonomous from model training to deployment.

4.4 MLflow as the cornerstone of the project

During this project, we heavily relied on MLflow, a platform designed to streamline the lifecycle of machine learning models. It allows for detailed tracking of various experiments, packaging of code to ensure reproducibility, and serving models to end-users. MLflow also features an API that is compatible with most machine learning libraries such as PyTorch, Scikit-learn, XGBoost, and supports multiple programming languages including Python, R, and Java. It incorporates various functionalities that facilitate the adoption of the MLOps approach.

There are numerous tools available for orchestrating tasks and data pipelines. Among the most popular (based on their GitHub stars) are Airflow, Luigi, Argo Workflows, Prefect, Kubeflow and BentoML. It's difficult to assert whether one is better to another; in reality, your choice largely depends on your IT infrastructure and project requirements. In our case, we opted to use MLflow for its ease of use and suitability for machine learning projects. Additionally, it is available in the SSP

Cloud catalog, simplifying its installation process and its connection with our MinIO storage where all the artifacts are automatically stored.

4.4.1 MLflow Projects

MLflow offers a format for packaging data science projects to promote code reuse and reproducibility. This format is simply called MLflow Project. Essentially, an MLflow project is nothing more than a directory containing the code and necessary resources (data, configuration files, etc.) for executing your project. It is summarized by an ‘MLproject‘ file listing the various commands to execute a pipeline as well as the necessary dependencies. In general, an MLflow project has the following structure:

```
Project_ML/
 ├─ artifacts/
 │   ├─ model.bin
 │   └─ train_text.txt
 ├─ code/
 │   ├─ main.py
 │   └─ preprocessing.py
 ├─ MLmodel
 ├─ conda.yaml
 ├─ python_env.yaml
 ├─ python_model.pkl
 └─ requirements.txt
```

4.4.2 MLflow Models

In addition to packaging a project, MLflow also allows you to package your model, regardless of the underlying machine learning library used[11] (among those compatible with MLflow, i.e., all the libraries you use!). Thus, two models trained with different libraries, say PyTorch and Keras, can be deployed and queried in the same way thanks to this layer added by MLflow. This harmonization becomes particularly valuable as you begin to accumulate various types of models, as it eliminates the need to modify your inference pipeline every time you introduce a new model type.

4.4.3 Tracking Server

MLflow provides a tracking server that includes both an ergonomic graphical interface and an API for logging various parameters, metrics, files, etc. during the training of your machine learning model. The tracking server is very useful for comparing

[11] You also have the capability to package our own custom model. This is precisely what we do to integrate our fastText model into MLflow.

the different experiments you have performed, storing them, and also being able to reproduce them. Indeed, each run stores the source of the utilized data along with the corresponding commit. When combined with MLflow Project, this capability enables the reproduction of every conducted run.

4.4.4 Model Registry

Once different experiments have been conducted and models that satisfy us have been selected, it is time to move on to the next step in the model lifecycle. Indeed, the chosen model must then be able to move into a production or pre-production environment. However, knowing the state of a model in its lifecycle requires very rigorous organization and is not so straightforward. MLflow has developed a feature that simplifies this version management of models through its Model Registry. This registry allows you to add tags and aliases to your models to define their position in their lifecycle and thus be able to retrieve them efficiently.

In general, a machine learning model goes through 4 stages that need to be known at all times:

1. **Experimental**
2. **Staging**
3. **Production**
4. **Archived**

4.4.5 MLflow in short

MLflow is an open-source project that provides a platform for tracking the lifecycle of a machine learning model from start to finish. It is not the only available tool, and it may not be the most suitable for some of your specific projects. However, we believe it offers several advantages, primarily its ease of use and its ability to meet the needs of the MLOps approach. It is important to keep in mind that this environment is still very new, and new open-source projects emerge every day, so it is necessary to stay up to date on the latest developments. In summary, MLflow allows you to:

- Simplify tracking the training of machine learning models through its API and tracking server.
- Integrate the main machine learning frameworks easily.
- Integrate your own framework if needed.
- Standardize your training script, enabling industrialization, for example, fine-tuning hyperparameters.
- Package your models for easy and harmonized querying across different frameworks.
- Store your models efficiently by assigning tags and facilitating lifecycle tracking.

4.5 Embracing the potential of Onyxia from training to deployment

To adhere closely to the MLOps approach, we strived to automate as much as possible the various stages of the machine learning model lifecycle. In the "automation" section of the SSP Cloud catalog, several services are available in addition to MLflow, including Argo CD and Argo Workflows, which we use daily for various applications.

Argo Workflows is an open-source container-native workflow engine designed for orchestrating parallel jobs on Kubernetes. It allows users to define complex multi-step workflows as code using YAML or JSON, simplifying the automation and management of tasks in a Kubernetes environment. The idea is to define workflows where each step of the process is a completely isolated container to ensure reproducibility. This allows for easy parallel execution of compute-intensive tasks for model training or data processing. In our case, we use it during the model training phase, particularly to optimize the various hyperparameters of our models. Instead of performing a grid search that would test all possible combinations in a single container, Argo Workflows creates as many containers as there are model combinations to train, with each container handling the training of a single model. Kubernetes optimizes resources within the cluster, and since we have configured MLflow in our code, all models and their metrics appear on the Tracking Server. It then becomes straightforward to compare the different trained models and select the best one using the comparison and visualization tools available on the UI. It is worth noting that it is relatively simple to make all these applications (Argo Workflows, MLflow, MinIO) communicate with each other with just a few configuration files because everything was well set up initially thanks to the Onyxia software. Without Onyxia, even the installation of these softwares can be challenging.

The use of tools like Argo Workflows is highly recommended for ML projects, firstly because they rely on technologies that promote reproducibility (containers), but also because they encourage viewing projects as a series of independent steps ranging from data retrieval to result dissemination via modeling. Adopting a pipeline-based approach to your machine learning or statistics projects is the first step toward greater reproducibility.

Once you have optimized, evaluated, and selected your best-performing model, it's crucial to make it available to other users. Unfortunately, this aspect is often overlooked by data scientists. A trivial way to share the model would be to provide your code and all necessary information to a third party to retrain your model on their own. Obviously, this approach is not optimal as it assumes that all users have the resources, infrastructure, and knowledge required for training. The goal is therefore to make your model available in a simple and efficient manner. The use of the SSP Cloud platform greatly simplifies this process and has also allowed us, as data scientists, to take control of our model until its deployment without relying on the IT team. Indeed, on the platform, it is possible to open services with Kubernetes administrator rights, enabling the deployment of applications. For making our machine learning model available, we opted to use a REST API. This seems to be the most suitable method in the vast majority of cases, as it meets several criteria:

- **Simplicity**: REST APIs provide an entry point that can hide the underlying complexity of the model, making its availability easier.
- **Standardization**: One of the main advantages of REST APIs is that they are based on the HTTP standard. This means they are language-agnostic, and requests can be made in XML, JSON, HTML, etc.
- **Modularity**: The client and server are independent. In other words, data storage, user interface, or model management are completely separated from the server.
- **Scalability**: The separation between the server and client allows REST APIs to be highly flexible and facilitate scalability. They can adapt to the load of concurrent requests.

To develop our API, we used the popular Python library FastAPI, which is relatively easy to use and comes with exhaustive documentation. The idea is to encapsulate all the API code and required software dependencies into a Docker image so that it can be deployed in a container on the Kubernetes cluster. Upon startup, the API will automatically retrieve the correct model from the MLflow model registry, which is located in a MinIO bucket. Using a MLflow model allows automatic integration of preprocessing before each prediction, regardless of the model framework used, greatly simplifying inference and ensuring streamlined code in the API. The model deployment process is summarized in Figure ??.

At this stage, we are already adhering to several MLOps principles in terms of reproducibility through containers and model versioning with MLflow. However, when it comes to automation, there is room for further improvement. Imagine you have deployed an API, and in the meantime, you have developed a new feature for it. For this new feature to be effectively integrated into your API, it is necessary to redeploy the new version of your API and remove the currently deployed one. The idea is to automate the creation of a new Docker image as soon as your API code is updated, and a container based on this latest image automatically deploys the new version of your API. To achieve this on the SSP Cloud, there is the ArgoCD service - CD for Continuous Deployment. ArgoCD continuously scans your Github repository, and as soon as it detects a change in the API code or in the version of the model to be used, it automatically deploys your API by communicating with the Kubernetes cluster. In reality, this process is a bit more nuanced. ArgoCD does not scan your entire codebase but only a few configuration files that indicate the versions of the Docker image to use and the model to deploy. To create a new version of the Docker image as soon as the API code is modified, we use the GitHub Actions feature of Github[12]. This deployment automation pipeline is depicted in Figure ??.

The autonomy of the data scientist up to the deployment of the model is particularly effective in terms of organization. Indeed, thanks to the deployment of an API or the transmission of a Docker image, transitioning the model to production is facilitated. On the IT side, querying an API is a common task, and no knowledge of machine learning is required to perform it.

[12] An equivalent feature also exists under Gitlab, called Gitlab CI.

4.6 Monitoring of the model

4.6.1 The importance monitoring

Once the modeling phase has been completed, along with training, optimization, and deployment of the model on a server to be accessible to users, it would be inaccurate to consider the work finished from the perspective of the data scientist. Traditionally, the responsibility of the data scientist is often limited to selecting the model to deploy, with the deployment task then delegated to data engineers. However, once in production, the model has not yet reached the end of its lifecycle, and it must be continuously monitored to prevent undesirable performance degradation.

The various components of MLOps, namely data (DataOps), models (ModelOps), and code (DevOps), add complexity to the lifecycle of a machine learning model by involving multiple stakeholders. Generally, three main actors are involved:

- Data scientists/data engineers
- IT teams
- Business teams

Sometimes, data scientists are integrated into business teams and data engineers into IT teams to foster better collaboration. This approach aims to streamline skills, unify terminologies, and align project objectives. In any cases, effective communication remains essential to ensure proper management of the lifecycle of a machine learning model, particularly concerning monitoring its behavior in a production environment. Continuous monitoring of the deployed model is extremely important to ensure the conformity of results to expectations, anticipate changes in data, and iteratively improve the model. The monitoring is a crucial element of the MLOps approach.

The concept of monitoring can take on different meanings depending on the context of the involved team. For IT teams, it primarily involves verifying the technical validity of the application, including aspects such as latency, memory consumption, or disk usage. Conversely, for data scientists or business teams, the focus is more on methodological monitoring of the model. However, real-time tracking of the performance of a machine learning model is often a complex task, given that ground truth is usually not known at the time of prediction. Therefore, it is common to use proxies to detect any signs of performance degradation. Two main types of degradation of a machine learning model are generally distinguished: data drift and concept drift.

- **Data drift**: Data drift occurs when the data used during inference in production exhibits significant differences compared to the data used during training, that is, $\mathbb{P}_{train}(X) \neq \mathbb{P}_{inference}(X)$.
- **Concept drift**: Concept drift is evoked when a change in the statistical relationship between the features (X) and the target variable (Y) is observed over time, that is, $\mathbb{P}_{train}(Y|X) \neq \mathbb{P}_{inference}(Y|X)$ and $\mathbb{P}_{train}(X) = \mathbb{P}_{inference}(X)$.

4.6.2 What do we monitor ?

In our use case, our objective is to achieve the highest number of correctly classified textual description while minimizing the number of textual description requiring manual intervention. Thus, our goal is to distinguish correct predictions from incorrect ones without prior access to ground truth. To accomplish this, we have developed a confidence index, which represents the difference between the two highest scores. Therefore, for a given textual description, if the confidence index exceeds a determined threshold, the textual description is automatically coded. Otherwise, the textual description is manually coded by an Insee's agent who has access to the five most probable codes according to the model to optimize manual intervention. The threshold beyond which a textual description is automatically coded constitutes a crucial parameter during production deployment, as it allows for arbitration between a high rate of automatic coding and lower model performance, and vice versa.

To monitor the behavior of our model in production, we have developed an interactive dashboard that enables visualization of several metrics of interest for the business teams. Among these metrics are the number of requests per day and the rate of automatic coding per day based on a given confidence index threshold. This visualization allows business teams to understand the rate of automatic coding they would have obtained if they had chosen different thresholds. It is also possible to visualize the distribution of obtained confidence indices and to compare two temporal windows to determine if there are changes in the distributions of predictions returned by the model[13].

Confidence indices can be analyzed at finer levels of granularity based on the aggregation level of the nomenclature, to determine which classes are most difficult to predict and which have more or less occurrences. Another part of the dashboard focuses more on the input data of the model and details, for example, the average number of words per textual description, the average number of sentences per textual description, as well as the most frequent words, etc.

4.6.3 How to implement a monitoring system in the SSP Cloud?

Although there are several solutions for monitoring machine learning models such as Evidently or Sifflet, we preferred to implement our own solution. Indeed, at the time of our trials, none of the open-source solutions available on the market seemed optimal to us. However, the field of monitoring machine learning models is still emerging and actively developing, so the statements made today may no longer be true at the time of reading these lines. Before opting, as we did, for an internal solution, we recommend exploring the available tools first.

In our case, we chose to use tools that we already mastered and that seemed suitable for our problem. We thus used DuckDB for optimized management of our

[13] To compare distributions, you need to calculate the distances between them using, for example the Bhattacharyya distance, the Kullback-Leibler divergence, the Hellinger distance or to perform statistical tests such as the Kolmogorov–Smirnov test, or the χ^2-test.

data and the Dashboard feature of Quarto for creating the interactive dashboard. First, to monitor in real-time the activities of your API, it is essential to integrate logs, whether to monitor technical or methodological aspects. We made the, perhaps debatable, decision to include in the API logs "business information" such as the code returned as output and the associated confidence index. The objective was to be able to build our dashboard based on the logs returned by the API. To exploit these logs, we set up a kind of Extract-Transform-Load (ETL) process in Python, which retrieves the API logs and transforms them into partitioned parquet files exploitable by the dashboard. Ideally, we would have preferred to rely on a tool capable of performing this parsing natively and more structured than us. Parseable could have been such a solution, but unfortunately, it was not installable on the SSP Cloud for technical reasons. We then use Quarto and its Dashboard feature to create an interactive dashboard. The various metrics present in the dashboard are calculated by making SQL queries directly on our parquet files with DuckDB. The dashboard is deployed in the same way as our API, that is, in a container based on an image that is automatically built via Github Actions as soon as the code associated with the dashboard is modified.

In summary, our monitoring pipeline consists of:

1. Writing "business logs" in the API for each request
2. With Argo Workflows, specifying a Cronjob that executes an ETL every night and produces a partitioned parquet file that it saves on MinIO
3. Continuously deployed with ArgoCD, the dashboard directly queries the parquet file using DuckDB, calculates the metrics, and displays them interactively

4.7 Continuous Annotation

In all machine learning projects, it is well-established that data is key, and in supervised machine learning problems, annotations play a crucial role in model performance. From the outset of our project, we were time-constrained and had to obtain results quickly, without having the time to set up an exhaustive annotation campaign. Consequently, we never had access to a truly annotated gold-standard test sample on which we could rely 100%. To evaluate the performance of our model, we relied on a subset of our training dataset (test/train split), knowing that its labeling was not perfect. After several months of model deployment, it became imperative to build such a gold-standard test sample. Firstly, this sample allows us to have an unbiased view of the model's performance in production on real data, particularly on data that has been automatically coded. Additionally, this sample enabled us to identify and correct inconsistencies present in our initial training dataset. Explaining the importance of annotation to the business teams was not an easy task, as they were reluctant to reallocate new human resources, requiring some expertise and potentially being tedious. The increase in automatic coding thanks to the model allowed for the reallocation of teams to annotation tasks.

Another reason that motivated the implementation of annotation campaigns is the redesign of the NACE nomenclature in 2025. From 2025 onwards, European statistical institutes will be required to use the latest version of NACE, namely NACE Rev. 2.1. This revision brings substantial changes that will require retraining a new model. For this, a new training dataset with the correct annotations is indispensable. A back-propagation work on the old training dataset is also considered to increase the size of the dataset. Thus, a dual annotation campaign was initiated in early 2024 on the SSP Cloud platform.

To carry out these annotation campaigns, we used the Label Studio service, available on the SSP Cloud. Label Studio is an open-source platform designed to facilitate data annotation. The first annotation campaign aims to create a test dataset allowing continuous evaluation of our model and improvement of the training dataset if necessary. To do this, we create a pool of text descriptions randomly sampled from the data passed through the API over the past three months. This sample is then sent to annotation by NACE experts using the UI of Label Studio. The annotation results are automatically saved on MinIO, transformed into parquet format, and then integrated into the monitoring dashboard to compute and observe various model performance metrics. The second annotation campaign is dedicated to the transition to the new version of NACE. The objective is to allocate certain experts to annotation in the new version of NACE to create a training dataset for the future model. In this revision, some codes are univocal, meaning for example that code 0112Z will become 0112Y, while others are multivocal, such as 0119Z, which can become either 0113Y or 0119Y. The challenge lies primarily with the latter, which is why, to streamline this annotation process, only the textual descriptions of multivocal codes are sent for annotation on Label Studio.

4.8 What next ?

We have demonstrated that an appropriate working infrastructure allows for the adoption of an MLOps approach for an entire ML project and the automation of as many steps as possible. The SSP Cloud is one of them. What is important to remember about the MLOps approach is that it is iterative: not everything can be implemented from the beginning, and time constraints can be limiting. Thus, it is advisable to start by implementing good development practices, and then to enrich your project according to its needs with increasingly more automation. On our part, we have identified several areas for improvement that we plan to implement soon. Firstly, we want to automate model retraining, for this we still need to improve our monitoring pipeline. Indeed, although the business teams have a dashboard allowing them to easily visualize the model's performance quasi-real-time (the dashboard is updated every night), there is not yet a system of alerts signaling when retraining is necessary and, more ambitiously, automatically triggering this retraining. Additionally, we plan to test other methodologies for our model, especially Transformer-type models, which could soon be usable on our production servers. Finally, although this does not directly optimize our pipeline in terms of MLOps, working on the revision of NACE will be an interesting challenge and will allow us to assess the relevance of the MLOps approach we have adopted since the beginning of the project.

4.9 Gouvernance d'un projet de ML/ challenges

On mets ça dans discussion ? ici dire tous les problèmes qu'on a eu (double preprocessing...) parler de limportance de la communication avec les gestionnaire pour la confiance dire que pour contrecarrer probleme de langage on a recruté un data scientist + les informaticien doivent se mettre à python

5 Discussion

5.1 Future

- Onyxia, un bien commun opensource largement réutilisé (Insee, SSB) → faciliter les contributions pour la postérité du projet open-source, qui dépasse l'Insee
- One-stop-shop : SSP Cloud comme plateforme de référence pour les projets de ML → croissance de l'offre de formation (+ traduction)
- Accompagner les réinstanciations (datafid, POCs dans le secteur privé)
- Multiplication des projets qui passent en prod (applications de dataviz, modèles de ML avec MLOps, webscraping : Jocas/WINs)

5.2 Discussion

- Cout d'entrée important pour l'organisation : stockage objet, cluster kube/conteneurisation
  - Choix fondamental d'archi → limite à la diffusion d'onyxia
  - Assumer le choix : compétences, organisation ...
  - Mais globalement : tendance favorable car beaucoup d'orga et INS font ce choix
- Cout d'entrée important pour le statisticien :
  - Non-persistence de l'environnement → git + stockage objet
  - Travail dans un conteneur → perte de repères sur l'environnement
  - Mais formation : bonnes pratiques + écoles de formation Insee + accompagnements
- SSP Cloud :

  - Instance ouverte → absence de données sensibles → grosse limitation des cas d'usage réalisables + frustrations → en résumé, difficile de maximiser à la fois innovation et sécurité (pb sur-contraint)
  - → résolution via le choix de l'innovation max car sujet des échanges inter-administration de données complexe + le SSP Cloud a pavé la voie à des instances internes, plus fermées → stratégie assumée "platform-as-a-package" : projet open-source packagé → facilité ++ de réinstanciation
  - Pas une plateforme de diffusion de données → pas de stratégie globale de gouvernance → le sujet de la méta-donnée n'est pas abordé.

- Gouvernance :

  - Quelle organisation ? Equipe DS centralisée qui vient en appui ou data scientists dans les orgas métiers ? Collaboration avec les équipes infos ? (cf. graphique orga/compétences de Romain)

Appendix

References

1. Daniel Abadi, Peter Boncz, Stavros Harizopoulos, Stratos Idreos, Samuel Madden, et al. The design and implementation of modern column-oriented database systems. *Foundations and Trends® in Databases*, 5(3):197–280, 2013.
2. Abdullah I Abdelaziz, Kent A Hanson, Charles E Gaber, and Todd A Lee. Optimizing large real-world data analysis with parquet files in r: A step-by-step tutorial. *Pharmacoepidemiology and Drug Safety*, 2023.
3. Afshin Ashofteh and Jorge M Bravo. Data science training for official statistics: A new scientific paradigm of information and knowledge development in national statistical systems. *Statistical Journal of the IAOS*, 37(3):771–789, 2021.
4. Ouafa Bentaleb, Adam SZ Belloum, Abderrazak Sebaa, and Aouaouche El-Maouhab. Containerization technologies: Taxonomies, applications and challenges. *The Journal of Supercomputing*, 78(1):1144–1181, 2022.
5. Thomas H Davenport and DJ Patil. Data scientist. *Harvard business review*, 90(5):70–76, 2012.
6. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
7. Pascaline Descy, Vladimir Kvetan, Albrecht Wirthmann, and Fernando Reis. Towards a shared infrastructure for online job advertisement data. *Statistical Journal of the IAOS*, 35(4):669–675, 2019.
8. DGINS. Bucharest memorandum on official statistics in a datafied society. `https://ec.europa.eu/eurostat/documents/13019146/13237859/\The+Bucharest+Memorandum+on+Trusted+Smart+Statistics+FINAL.pdf/7a8f6a8f-9805-e77c-a409-eb55a2b36bce?t=1634144384767`, 2018.
9. Bijesh Dhyani and Anurag Barthwal. Big data analytics using hadoop. *International Journal of Computer Applications*, 108(12):0975–8887, 2014.
10. EUROSTAT. Essnet big data 2 - final technical report. `https://wayback.archive-it.org/12090/20221110013641/https://ec.europa.eu/eurostat/cros/system/files/\wpa_deliverable_a5_final_technical_report_2021_06_29.pdf`, 2021.
11. Apache Software Foundation. Apache parquet. `https://parquet.apache.org/`, 2013.

12. Apache Software Foundation. Apache arrow. `https://arrow.apache.org/`, 2016.
13. Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 29–43, 2003.
14. Taeke Gjaltema. High-level group for the modernisation of official statistics (hlg-mos) of the united nations economic commission for europe. *Statistical Journal of the IAOS*, 38(3):917–922, 2022.
15. Shivani Gokhale, Reetika Poosarla, Sanjeevani Tikar, Swapnali Gunjawate, Aparna Hajare, Shilpa Deshpande, Sourabh Gupta, and Kanchan Karve. Creating helm charts to ease deployment of enterprise application and its related services in kubernetes. In *2021 international conference on computing, communication and green engineering (CCGE)*, pages 1–5. IEEE, 2021.
16. Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.
17. Alexander Kowarik and Magdalena Six. Quality guidelines for the acquisition and usage of big data with additional insights on web data. In *4th International Conference on Advanced Research Methods and Analytics (CARMA 2022)*, pages 269–269. Editorial Universitat Politècnica de València, 2022.
18. Dominik Kreuzberger, Niklas Kühl, and Sebastian Hirschl. Machine learning operations (mlops): Overview, definition, and architecture. *IEEE Access*, 2023.
19. Marie Leclair et al. Utiliser les données de caisses pour le calcul de l'indice des prix à la consommation. *Courrier des statistiques*, 3:61–75, 2019.
20. Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. A survey of devops concepts and challenges. *ACM Computing Surveys (CSUR)*, 52(6):1–35, 2019.
21. Yun Li, Manzhu Yu, Mengchao Xu, Jingchao Yang, Dexuan Sha, Qian Liu, and Chaowei Yang. Big data and cloud computing. *Manual of digital earth*, pages 325–355, 2020.
22. Ling Liu. Computing infrastructure for big data processing. *Frontiers of Computer Science*, 7:165–170, 2013.
23. Sybille Luhmann, Jacopo Grazzini, Fabio Ricciato, Mátyás Mészáros, Jean-Marc Museux, and Martina Hahn. Promoting reproducibility-by-design in statistical offices. In *2019 New Techniques and Technologies for Statistics (NTTS) conference*, 03 2019.
24. Marcia McNutt. Reproducibility. *Science*, 343(6168):229–229, 2014.
25. Mike Mesnier, Gregory R Ganger, and Erik Riedel. Object-based storage. *IEEE Communications Magazine*, 41(8):84–90, 2003.
26. David Moreau, Kristina Wiebels, and Carl Boettiger. Containers for computational reproducibility. *Nature Reviews Methods Primers*, 3(1):50, 2023.
27. Justice Opara-Martins, M Sahandi, and Feng Tian. A holistic decision framework to avoid vendor lock-in for cloud saas migration. *Computer and Information Science*, 10(3), 2017.
28. Justice Opara-Martins, Reza Sahandi, and Feng Tian. Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective. *Journal of Cloud Computing*, 5:1–18, 2016.
29. Mark Raasveldt and Hannes Mühleisen. Duckdb: an embeddable analytical database. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1981–1984, 2019.
30. Fabio Ricciato, Freddy De Meersman, Albrecht Wirthmann, Gerdy Seynaeve, and Michail Skaliotis. Processing of mobile network operator data for official statistics: the case for public-private partnerships. In *104th DGINS conference*, 2018.
31. Fabio Ricciato, Albrecht Wirthmann, Konstantinos Giannakouris, Michail Skaliotis, et al. Trusted smart statistics: Motivations and principles. *Statistical Journal of the IAOS*, 35(4):589–603, 2019.
32. Anam Saiyeda and Mansoor Ahmad Mir. Cloud computing for deep learning analytics: A survey of current trends and challenges. *International Journal of Advanced Research in Computer Science*, 8(2), 2017.

33. David Salgado, Luis Sanguiao-Sande, Sandra Barragán, Bogdan Oancea, and Milena Suarez-Castillo. A proposed production framework with mobile network data. In *ESSnet Big Data II - Workpackage I - Mobile Network Data*, 2020.
34. S Samundiswary and Nilma M Dongre. Object storage architecture in cloud for unstructured data. In *2017 International Conference on Inventive Systems and Control (ICISC)*, pages 1–6. IEEE, 2017.
35. Steven Vale. International collaboration to understand the relevance of big data for official statistics. *Statistical Journal of the IAOS*, 31(2):159–163, 2015.
36. Rafael Vaño, Ignacio Lacalle, Piotr Sowiński, Raúl S-Julián, and Carlos E Palau. Cloud-native workload orchestration at the edge: A deployment review and future directions. *Sensors*, 23(4):2215, 2023.
37. Qi Zhang, Ling Liu, Calton Pu, Qiwei Dou, Liren Wu, and Wei Zhou. A comparative study of containers and virtual machines in big data environment. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 178–185. IEEE, 2018.