DataOps
Good Practices for Data Science

Oscar Romero, Anna Queralt and Marc Maynou

DTIM RESEARCH GROUP (http://www.essi.upc.edu/dtim/)

UNIVERSITAT POLITÈCNICA DE CATALUNYA - BARCELONATECH

Data Science Projects

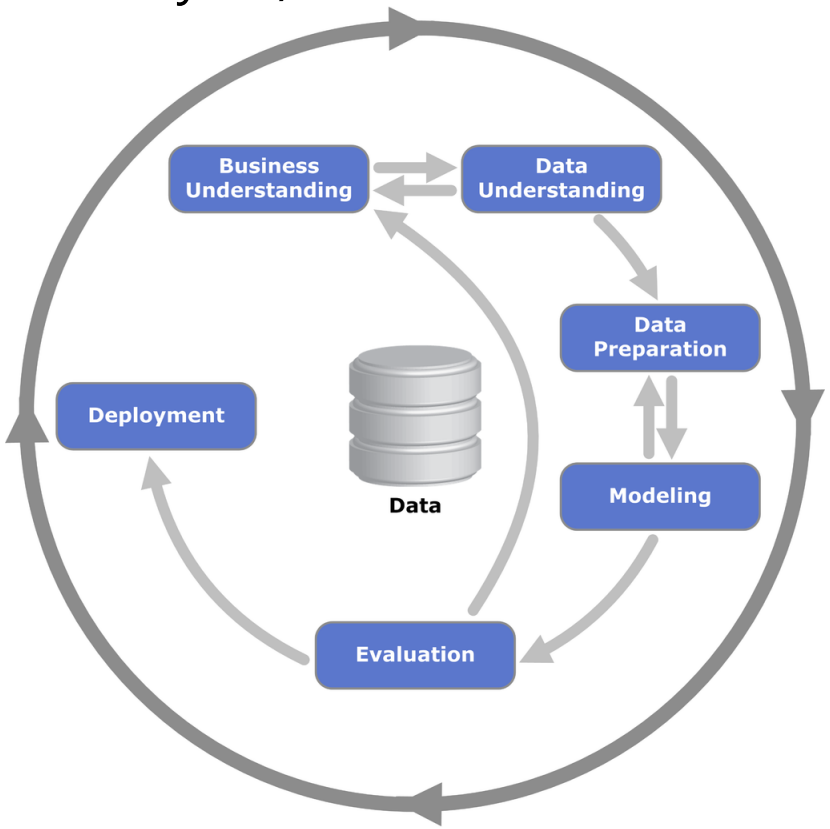
Data science projects require creating systems that deploy data pipelines spanning three different areas:

- Business understanding (domain)
  - What do we want to analyse?
  - What is the added value of an analytical question for the organisation?
- Data management
  - Data discovery
  - Data modeling
  - Data storage
  - Data processing
  - Data querying
- Data analysis
  - Data preparation
  - Modeling
  - Validation
  - Explainability
  - Visualization

CRISP-DM Methodology for Data Science Projects

Stands for CRoss-Industry Standard Process for Data Mining

- An open standard process model describing common approaches by data mining experts (or in general, data analysts)



CRISP-DM Criticism

It is too high level

- It simply identifies the main steps and separates concerns but it does not determine good practices

It is not generic enough

- Data engineering aspects ignored. It does not set grounds for reusability of processes at the organisation level

In general, CRISP-DM does not prevent bad practices and it is too high level as to provide value to practitioners

Exercise 1: Looking for clusters visually

From the course *Transition to Data Science*. Buy the entire course for just $10 for many more exercises and helpful video lectures.

You are given an array `points` of size 300x2, where each row gives the (x, y) co-ordinates of a point on a map. Make a scatter plot of these points, and use the scatter plot to guess how many clusters there are.

Step 1: Load the dataset *(written for you)*.

In [1]:
```
import pandas as pd

df = pd.read_csv('../datasets/ch1ex1.csv')
points = df.values
```

Step 2: Import PyPlot

In [2]:
```
import matplotlib.pyplot as plt
```

Step 3: Create an array called `xs` that contains the values of `points[:,0]` - that is, column `0` of `points`.

In [3]:
```
xs = points[:,0]
```

Step 3: Create an array called `ys` that contains the values of `points[:,1]` - that is, column `1` of `points`.

In [4]:
```
ys = points[:,1]
```

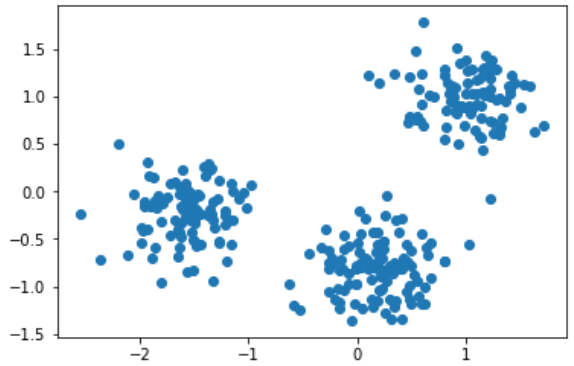
Step 4: Make a scatter plot by passing `xs` and `ys` to the `plt.scatter()` function.

In [5]:
```
plt.scatter(xs, ys)
```

Out[5]:
```
<matplotlib.collections.PathCollection at 0x110fc45c0>
```

Step 5: Call the `plt.show()` function to show your plot.

In [6]:
```
plt.show()
```



Read data from temporal files

Use off-the-shelf ML libraries

Pre-process the data to fit the needs of the algorithm input parameters

Run the algorithm

Visualise the result to allow its interpretation

Extracted from: https://github.com/benjaminwilson/python-clustering-exercises/

The Baseline Pipeline

Pros

- Dynamic and integrated environment that includes text, code and visualisations
- Enables ad-hoc (not systematic) sharing of analytical workflows
- Enables trial / error (the data science loop)
- Access to tones of analytical libraries

Cons

- Most data science libraries process data locally (e.g, in CSV). This approach compromises:
  - Persistence
  - Efficiency in data access
  - Concurrency
  - Reliability
  - Security
  - Performance
  - Governance

NO COMMON BACKBONE

- No code / data sharing / reusage
- No single source of truth (data / code)
- No global optimizations

Operationalizing (and Automating) Data Science Pipelines

MLOps / DataOps

Main Challenge

"Quality deliveries, with short cycle time, need a high degree of automation"

Ebert, C., Gallardo, G., Hernantes, J., Serrano, N. (2016). DevOps. IEEE Software 33(3): 94-100.

Main Challenge

"Quality deliveries, with short cycle time, need a high degree of automation"

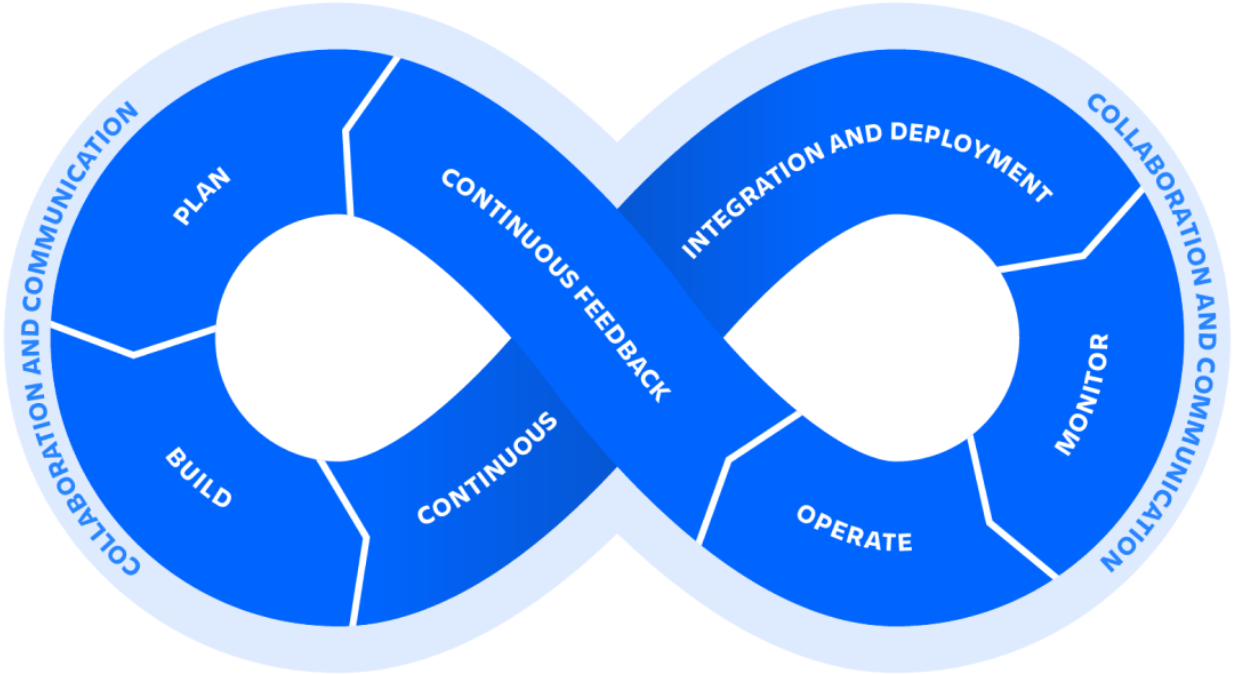
Ebert, C., Gallardo, G., Hernantes, J., Serrano, N. (2016). DevOps. IEEE Software 33(3): 94-100.

DevOps

DevOps is about fast, flexible development and provisioning processes.

It integrates the two worlds of development and operation, using automated development, deployment, and infrastructure monitoring. Its an organizational shift in which, instead of distributed isolated groups performing functions separately, cross-functional teams work on continuous operational feature deliveries.

Ebert, C., Gallardo, G., Hernantes, J., Serrano, N. (2016). DevOps. *IEEE Software 33*(3): 94-100.



The DevOps Lifecycle

- Plan: elicit requirements and prioritize new functionalities to be implemented in iterations. To improve speed and quality, agile methodologies must be applied
- Build: the process transforming standalone code into software artifacts. Build processes include version control, documentation, managing dependencies, code quality, testing and compilation. Build also considers deploying an application to different environments: typically, development (*dev*), testing (*test*) and production (*prod*)
- Continuous integration (CI) and delivery: developers' work must be integrated as often as possible to facilitate testing and reduce time-to-market. CI pipelines orchestrates automated builds, tests and deployment into release workflows
- Monitoring: of the infrastructure and the deployed software in production
- Continuous feedback: monitoring results loop back into the next plan iteration and should be considered to plan the ahead iterations

Tools for DevOps

- Agile planning: Jira, Trello, ActiveCollab, Slack / Discord (team communication), etc.
- Build:
  - Dependencies management: Ant, Maven, Gradle (Java) / pip (Python) / SBT (Scala), etc.
  - Code versioning: Git, etc.
  - Code quality: SonarCloud, etc.
  - BDD Testing: Selenium, Cucumber, etc.
  - TDD Testing: Junit, Mockito (Java) / Pytest, Pydantic (Python), etc.
- Continuous integration and delivery: Jenkins, Gitlab, IBM ModelOps, AWS Integration, CI Google Cloud, etc.
- Monitoring: Nagios (infrastructure), Prometheus, VisualVM (software profiling), Icinga (monitoring software calls), etc.
- Continuous feedback: same tools as in agile planning

From DevOps to DataOps / MLOps

DevOps ignores a key aspect in Data Science: data and its lifecycle

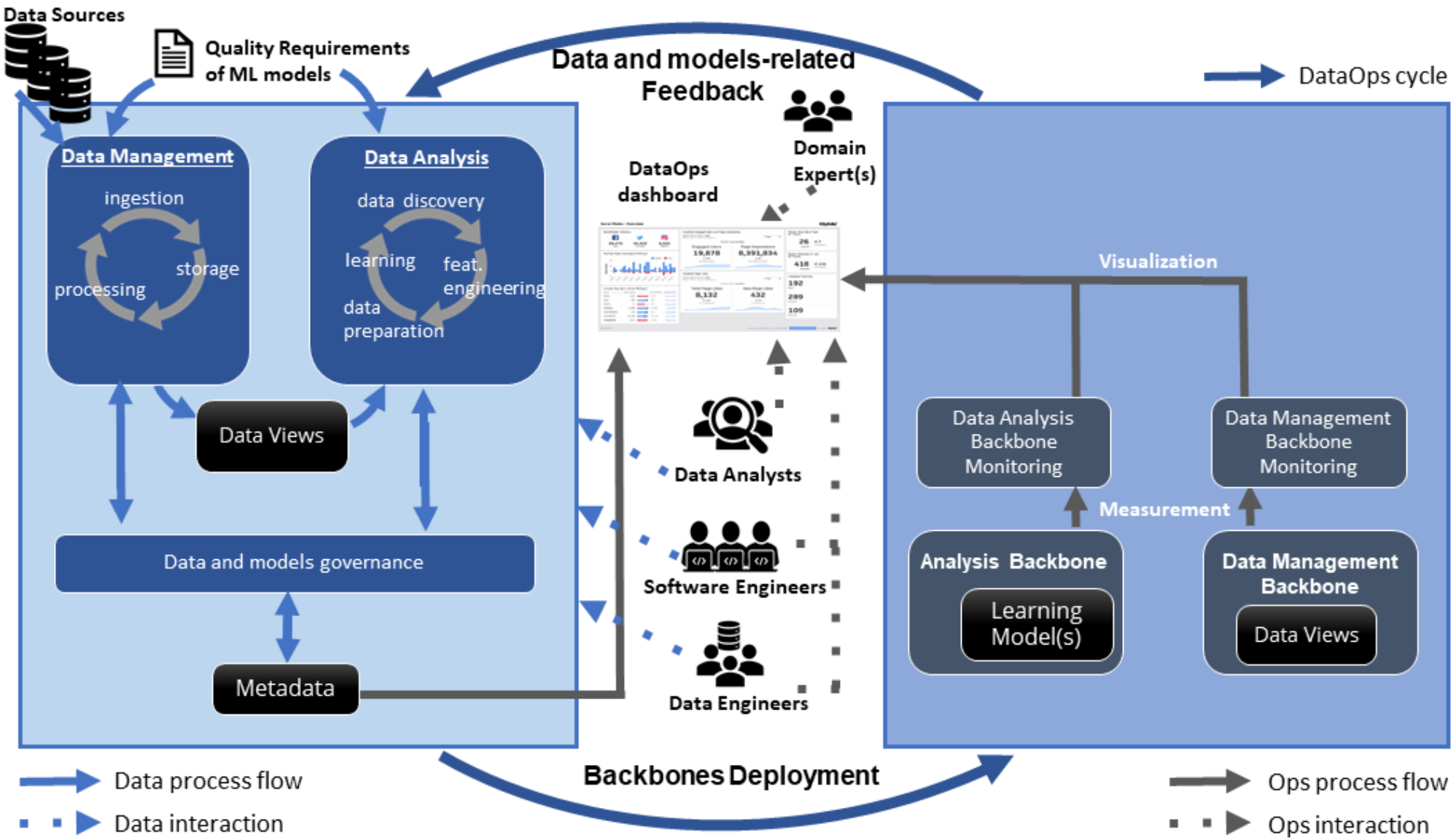
DataOps was introduced to cover this gap by *combining an integrated and process-oriented perspective on data with automation and methods from agile software engineering, like DevOps, to improve quality, speed, and collaboration and promote a culture of continuous improvement.*

Ereth, J. (2018). DataOps - Towards a Definition. LWDA 2018: 104-112.

In short, the DataOps lifecycle is needed to manage the complexity of the data lifecycle in data science projects (data engineering)

***Disclaimer**: you may read about MLOps too. However, in essence, DataOps / MLOps talk about the same problem. Simply, the latter focuses more on the ML part while the former cover the whole data lifecycle. We aim at providing a holistic view in this course and that is why we choose DataOps in front of MLOps*

DataOps in a Nutshell



DataOps in a Nutshell

The data management backbone (common for the whole organisation)

- Ingest and store external data into the system
  - Data Integration (standardization and data crossing)
    - Syntactic homogenization of data
    - Semantic homogenization of data
  - Clean data / eliminate duplicates
- Expose a cleaned and centralized repository
  - Data profiling

The data analysis backbone (repeats for every analytical pipeline)

- Extract a data view from the centralized repository
- Feature engineering
- Specific pre-processing for the given analytical task at hand
  - Labelling
  - Data preparation specific for the algorithm chosen
- Create training and validation datasets
- Learn models (either descriptive statistical analysis or advanced predictive models)
- Validate and interpret the model

DataOps in a Nutshell

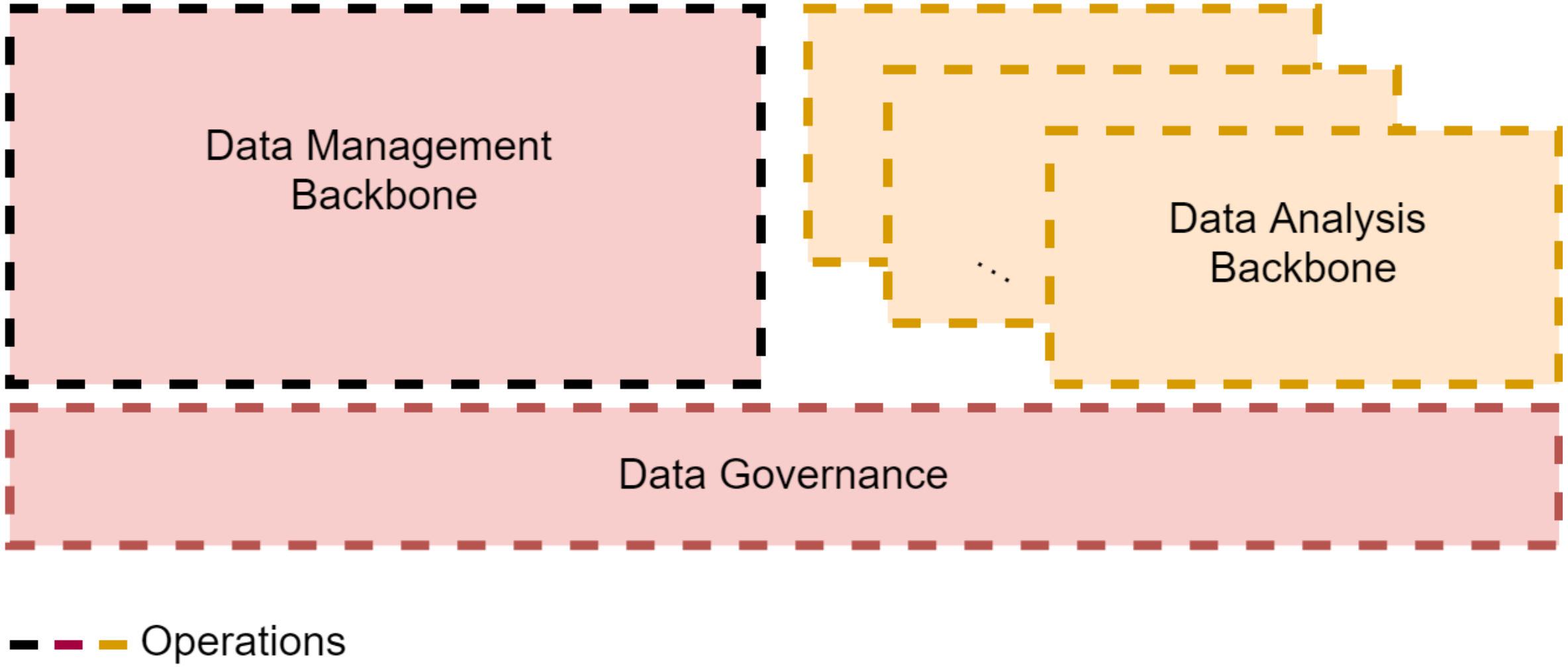
Operations (common per project)

- Data management backbone monitoring: metrics such as processes performance, disk and memory usage, etc.
- Data analysis backbone monitoring: metrics such as model accuracy, training time, fairness, etc.

Governance (common for the whole organisation)

- Domain-specific metadata: information about the day-by-day vocabulary used by analysts and their link to data variables / data sources, data preparation tasks for a given analysis, features and datasets used by the trained models, etc.
- Domain-independent metadata: information about domain-independent processes conducted on data such data lineage / traceability, etc.
- Physical metadata: information required to automatically process physical records such as data formats / data types, format-specific templates, etc.

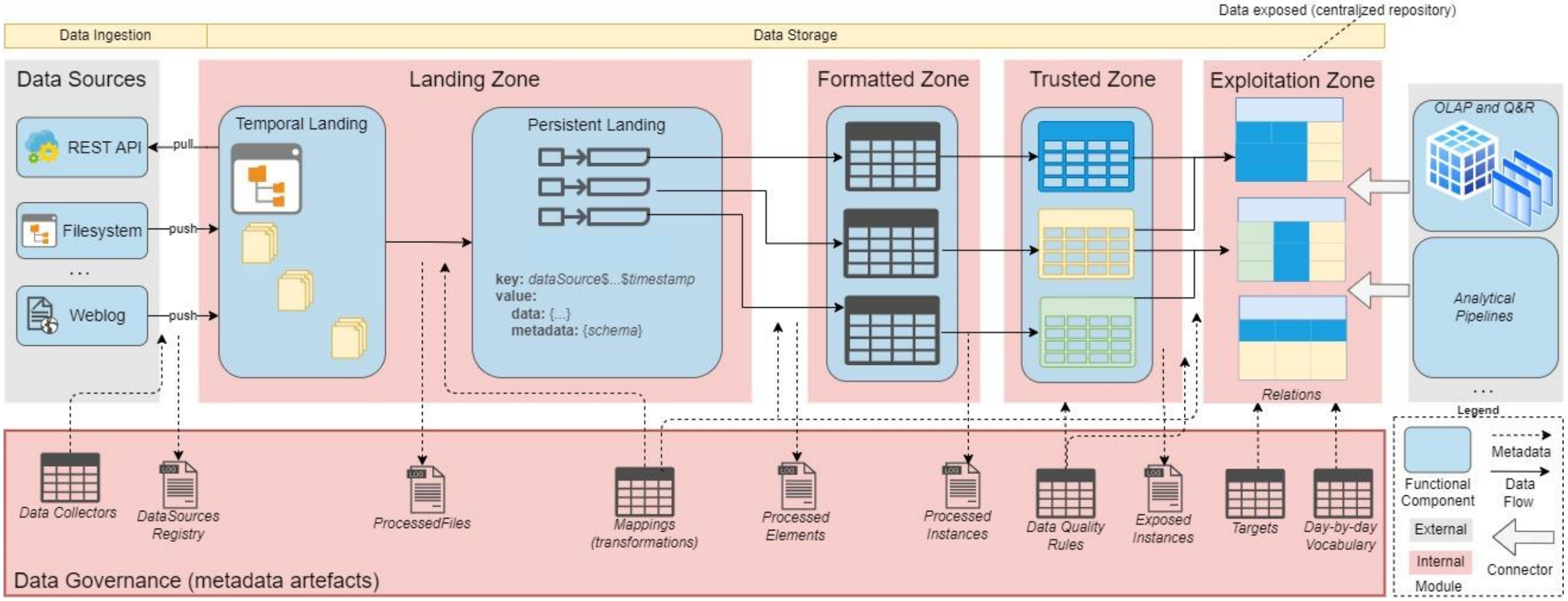
DataOps in a Nutshell



The Data Management Backbone

And its Governance

The Data Management Backbone



The Data Management Backbone

- Data is ingested in the landing zone as it is produced (raw data)
  - The temporal landing stores temporary the files, until they are processed
  - The persistent landing tracks ingested files by data source and timestamp (i.e., versions)
- Data is then homogenized, according to a canonical data model in the formatted zone (syntactic homogenization)
- The trusted zone is where data cleaning happens. However, only generic data quality techniques (i.e., independent of a specific project) occur here
- The exploitation zone exposes data ready to be consumed / analysed either by advanced analytical pipelines or external tools. Two main kind of tasks are conducted to generate this zone (semantic homogenization):
  - Data integration: new data views are generated by combining the instances from the trusted zone. A view may serve several data analysis.
  - Data quality processes (after integration): data quality required to have a wider view (e.g., instances from different sources or requiring to happen after data Integration is conducted)

Governing the Data Management Backbone

The data governance layer must fulfill the following tasks:

- Monitor all the data flow executions between zones, as well as the required metadata to allow their incremental execution (logs)
- Metadata artifacts capturing the schema of the data views created in the exploitation zone, but also in any of the intermediate zones
- Metadata required to automate processes: templates to process different types of data, database connections, etc.
- Mappings between zones: transformations conducted between elements of two consecutive zones to enable data lineage / traceability
- Profiling information of every zone: for maintenance purposes for the landing, trusted and formatted zones and analytical purpose in the exploitation zone

The Relevance of the Exploitation Zone

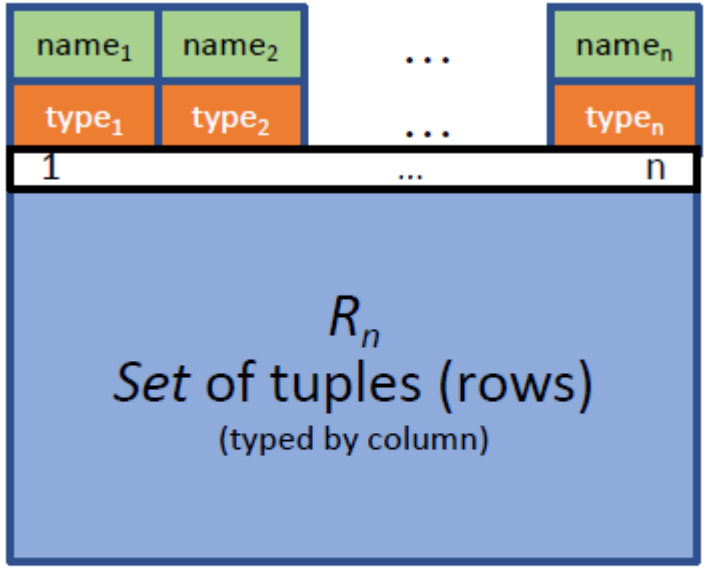
It is the zone where data is exposed either to in-house data scientists (to conduct ad-hoc advanced data analysis tasks) or to external tools

- Query & reporting (descriptive data analysis): relations (traditional OLAP engines over relational databases)
  - Seen in Data Warehousing
- Machine Learning and Data Mining (predictive data analysis): they require developing data analysis pipelines. These pipelines entail first transforming data into dataframes (e.g., R, Python libraries / frameworks such as Pandas, SAS, etc.) and conducting all the steps for learning models. Dataframes are nowadays the standard de facto for data analysis, also for distributed Machine Learning and Data Mining (e.g., MLlib). Deep Learning frameworks, instead, require tensors.
  - Seen in the 2nd part of the project

Canonical model: Relational

Follow the relational model data structure

- Schema of the relation
  - Name of the relation
  - Set of attributes: each attribute has a name and a predefined domain (datatype)
- Range of the relation
  - A tuple is an element of the range meeting the schema (NULL values allowed)
- Integrity constraints:
  - Entity constraint, PKs, FKs, CHECKs, etc.



Relation
(Table)

Manipulation language: relational algebra (SQL)

Tools for Data Science

Methodology

According to good practices, we must differentiate two types of environments: development and operations

- Development: during development, the most popular IDE nowadays are notebooks (e.g., Jupyter, Zeppelin, RStudio). Their dynamicity is key to enable trial and error and fast prototyping (e.g., decide the right system architecture, database schemas, etc.)
- Operations: in operations, notebooks are not used. Instead, continuous integration and deployment tools are used (e.g., Gitlab, Jenkins, etc.), following the good practices set by DevOps.

Therefore, realise that in most Data Science projects development (the glue code putting all the previously mentioned software pieces together) is coded in Python (or any other language enabling fast prototyping and coding iterations) via notebooks, while operations is implemented with a robust language (e.g., Java) via continuous integration and development tools.

Tools for the Data Management Backbone

- Ingestion: it can be performed by means of off-the-shelf tools or ad-hoc scripting. Further, it can be pull or push
  - Scripting: Java, Python, bash scripts, etc.
  - Tools: Kafka, Spark, etc.
- Storage: when facing Big Data, distributed storage is a must. Anyway, specially in the exploitation zone, a vast range of tools might be used to store a given data view
  - Distributed storage: distributed file Systems (e.g., HDFS) and distributed databases (e.g., HBase, SimpleDB, Cassandra).
  - Centralized storage: relational analytical databases (e.g., CockroachDB, DuckDB), etc.
- Data quality and profiling: in the past, they were tightly related to the analytical tools (e.g., SAP, IBM, etc.). Nowadays, there are two trends:
  - An extensive offer of stand-alone tools that fit better the required *openness* view of data science: Trifacta, HoloClean, Attacama, Informatica, Dataprep.ai, etc.
  - Libraries provided by the most popular data analysis libraries (e.g., Pandas, Keras, Pytorch, etc.)
- Transformations between zones: Spark and Flink the most used ones. There is the option of going to a lower level of abstraction and use frameworks that allow to tailor the execution to your needs and gain efficiency: e.g., Akka.

Main Programming Languages

- Development environment
  - An exploratory phase where data scientists aim to flexibly explore the data sources at hand, select the most relevant variables or evaluate different models.
  - Desired features
    - Dynamically typed languages
    - Interpreted code (no compilation required)
    - Unnecessary verbose code
  - Exemplary languages: Python, R, Perl
- Operations environment
  - A consolidation phase where the goal is to deploy a data infrastructure with performance guarantees and robust to external changes.
  - Desired features
    - Statically typed languages
    - Compiled code (low-level optimization and memory management)
    - Built-in parallelism for concurrent applications
    - Modularity
    - Advanced error and exception management
  - Exemplary languages: Java, Scala, C++