# Machine Learning Data Lifecycle in Production

MLOps Specialization Course 2

In the second course of Machine Learning Engineering for Production Specialization, you will build data pipelines by gathering, cleaning, and validating datasets and assessing data quality; implement feature engineering, transformation, and selection with TensorFlow Extended and get the most predictive power out of your data; and establish the data lifecycle by leveraging data lineage and provenance metadata tools and follow data evolution with enterprise data schemas.

Understanding machine learning and deep learning concepts is essential, but if you're looking to build an effective AI career, you need production engineering capabilities as well. Machine learning engineering for production combines the foundational concepts of machine learning with the functional expertise of modern software development and engineering roles to help you develop production-ready skills.

## Week 3: Data Journey and Data Storage

- Understand the data journey over a production system's lifecycle and leverage ML metadata and enterprise schemas to address quickly evolving data.
- Describe data journey through data lineage and provenance
- Integrate the sequence of pipeline artifacts into metadata storage using ML Metadata library
- Iteratively create enterprise data schema
- Explain how to integrate enterprise data into feature stores, data warehouses and data lakes

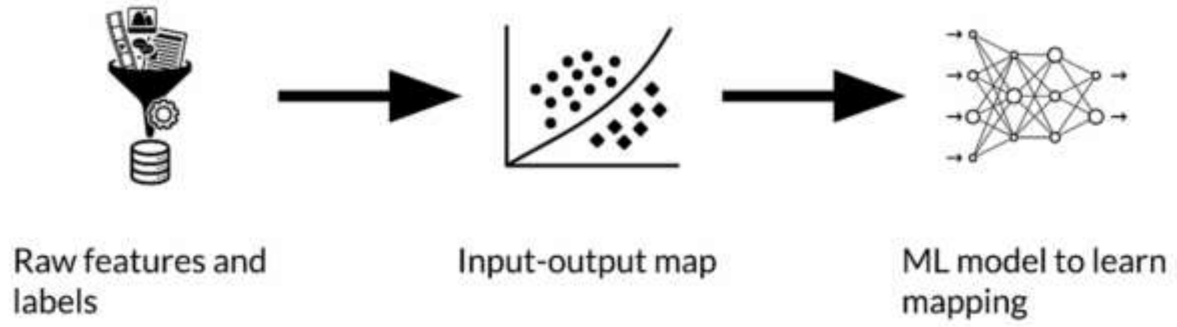
| | |
|---|---|
| **Compiled By** | **Peter Boshra** |
| **LinkedIn** | **https://www.linkedin.com/in/peterboshra/** |
| **GitHub** | **https://github.com/PeterBushra** |
| **Email** | **Dev.PeterBoshra@gmail.com** |

## 2.3 Data Journey and Data Storage

### Outline

- The data journey
- Accounting for data and model evolution
- Intro to ML metadata
- Using ML metadata to track changes

### The data journey



## Data transformation

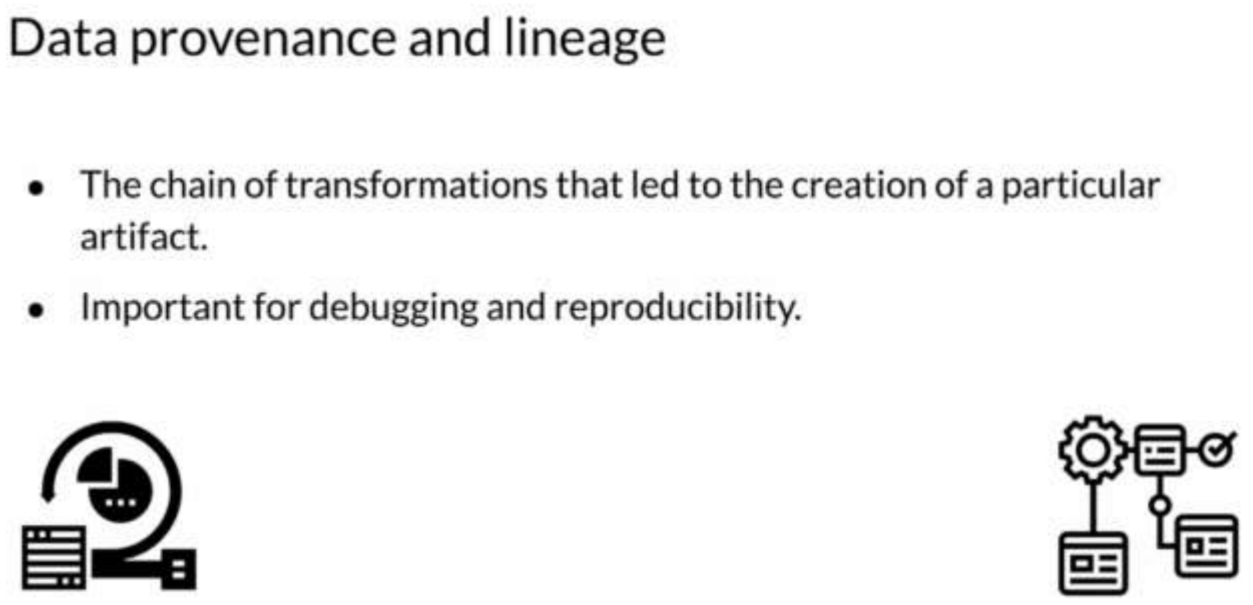
- Data transforms as it flows through the process
- Interpreting model results requires understanding data transformation

## Artifacts and the ML pipeline

Scoping
Data
Modeling
Deployment

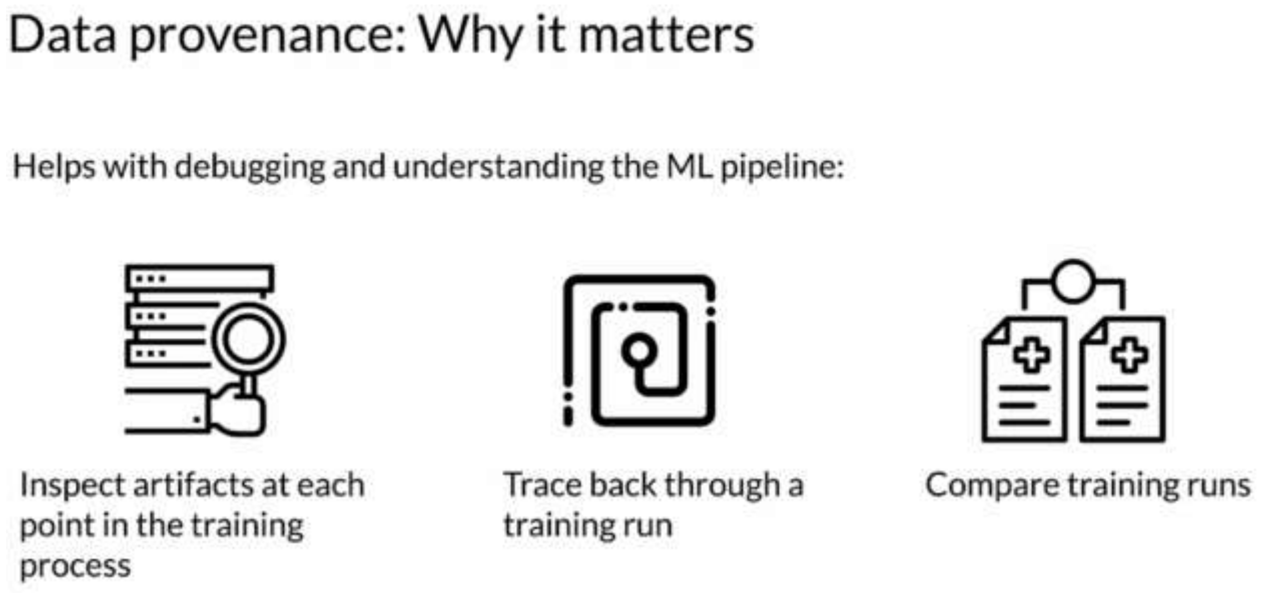
- Artifacts are created as the components of the ML pipeline execute
- Artifacts include all of the data and objects which are produced by the pipeline components
- This includes the data, in different stages of transformation, the schema, the model itself, metrics, etc.

What exactly is an artifact? Each time a component produces a result, it generates an artifact. This includes basically everything that is produced by the pipeline, including the data in different stages of transformation, often as a result of feature engineering and the model itself and things like the schema, and metrics and so forth. Basically, everything that's produced, every result that is produced as an artifact.



## Data provenance and lineage

- The chain of transformations that led to the creation of a particular artifact.
- Important for debugging and reproducibility.



## Data provenance: Why it matters

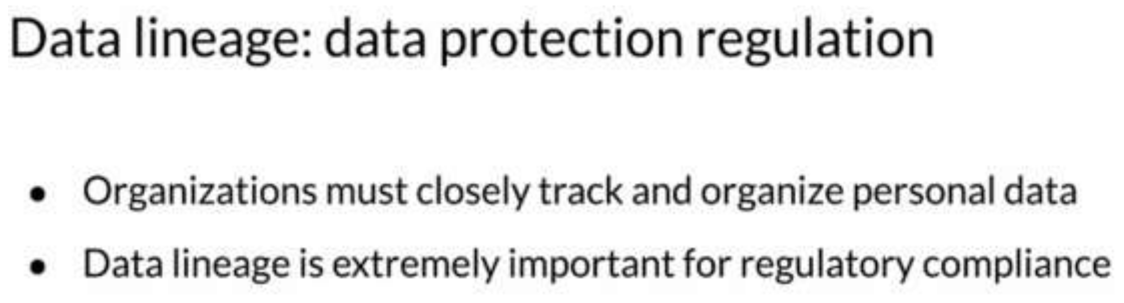
Helps with debugging and understanding the ML pipeline:

Inspect artifacts at each point in the training process

Trace back through a training run

Compare training runs

Data provenance or lineage is a sequence of the artifacts that are created as we move through the pipeline. Those artifacts are associated with a code and components that we create. Tracking those sequences is really key for debugging and understanding the training process and comparing different training runs that may happen months apart. Data provenance matters a great deal and it helps us to understand the pipeline and to perform debugging. Debugging and understanding requires inspecting those artifacts at each point in the training process, which can help us understand how those artifacts were created and what the results actually mean. Provenance will also allow you to track back through a training run from any point in the process. Also, provenance makes it possible to compare training runs, and understand why they produce different results. Under GDPR or the general data protection regulation, organizations are accountable for the origin, changes and location of personal data. Personal data is highly sensitive, so tracking the origins and changes along the pipeline are key for compliance. Data lineage is a great way for businesses and organizations to quickly determined how the Data has been used and which Transformations were performed as the Data moved through the pipeline. Data provenance is key to interpreting model results. Model understanding is related to this, but it's only part of the picture. The model itself is an expression of the data in the training set. In some sense, we can look at the model as a transformation of the Data. Provenance also helps us understand how the model evolves as it is trained and perhaps optimized. Let's add an important ingredient here, tracking different Data versions. Managing a data pipelines is a big challenge as data evolves through the natural life cycle of a project, over many different training runs. A Machine learning when it's done properly, should produce results that can be reproduced fairly consistently. There will naturally be some variance, but the results should be closed.



## Data lineage: data protection regulation

- Organizations must closely track and organize personal data
- Data lineage is extremely important for regulatory compliance

## Data provenance: Interpreting results

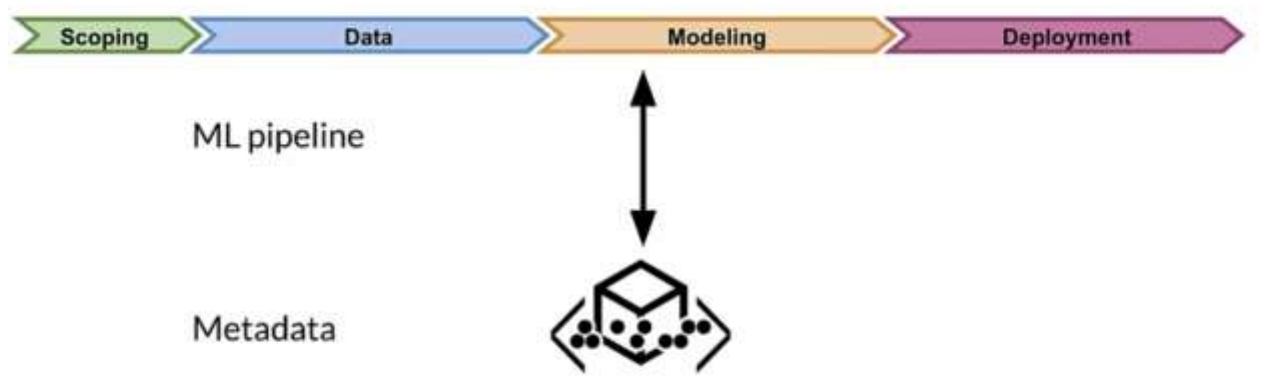
Data transformations sequence leading to predictions

Understanding the model as it evolves through runs

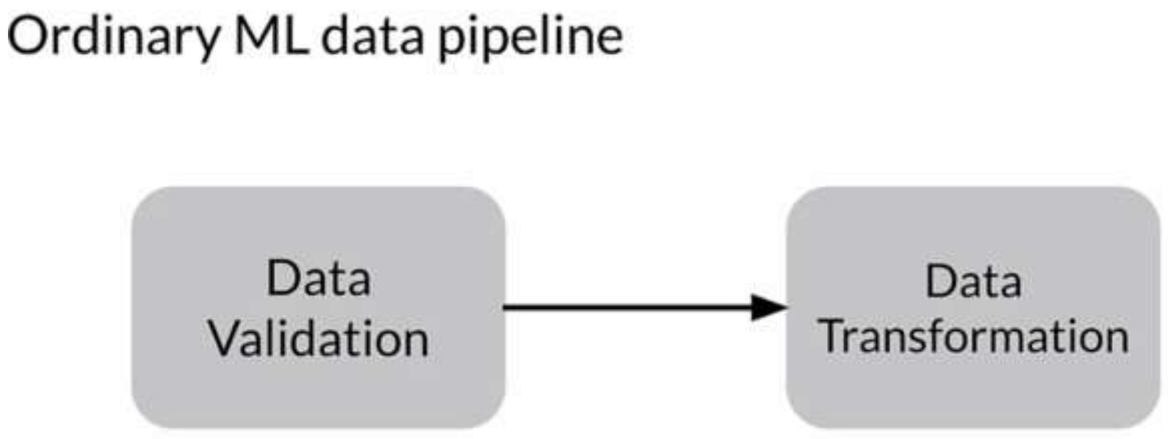
## Data versioning

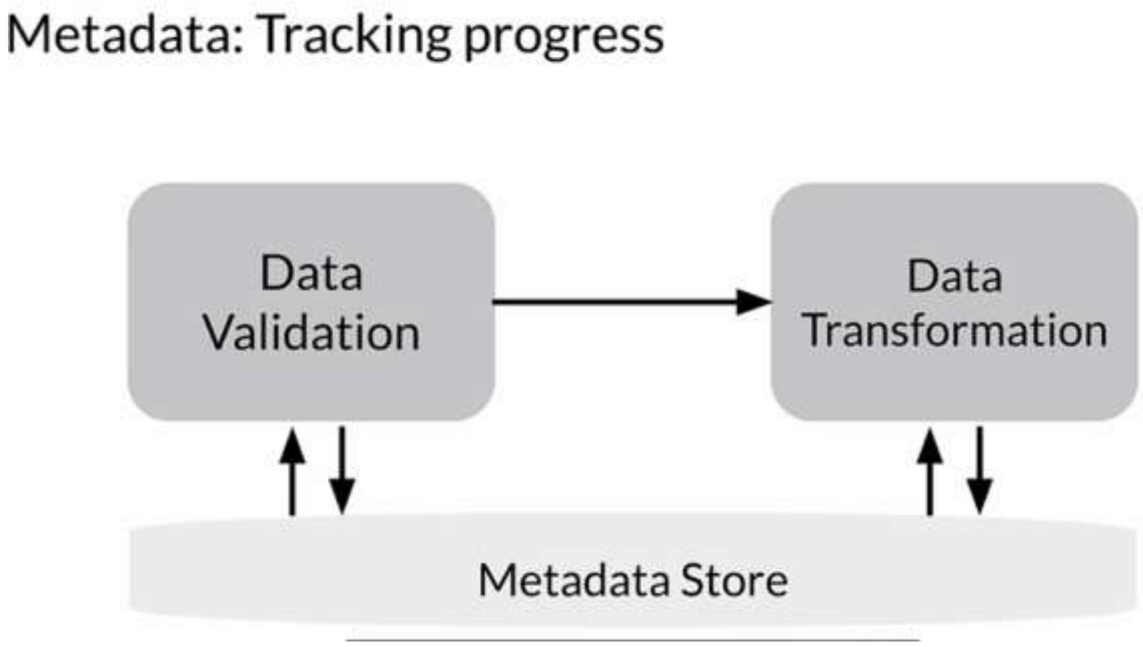
- Data pipeline management is a major challenge
- Machine learning requires reproducibility
- Code versioning: GitHub and similar code repositories
- Environment versioning: Docker, Terraform, and similar
- Data versioning:
    - Version control of datasets
    - Examples: DVC, Git-LFS

## Metadata: Tracking artifacts and pipeline changes



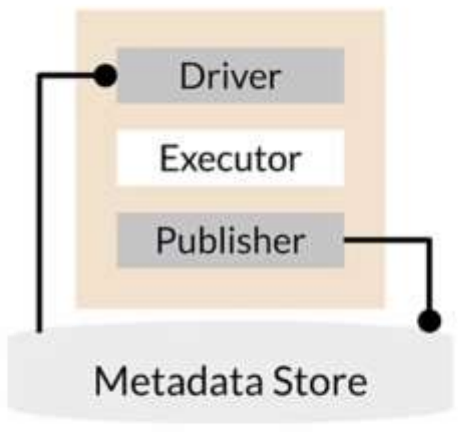
ML metadata or MLMD can help you with tracking artifacts and pipeline changes during a production life cycle. Every run of a production ML pipeline generates metadata containing information about the various pipeline components and their executions or training runs and the resulting artifacts. For example, trained models. In the event of unexpected pipeline behavior or errors, this metadata can be leveraged to analyze the lineage of pipeline components and to help you debug issues. Think of this metadata as the equivalent of logging in software development. MLMD helps you understand and analyze all the interconnected parts of your ML pipeline, instead of analyzing them in isolation. Now consider the two stages in ML engineering that you've seen so far. First you've done data validation, then you've passed the results onto data transformation or feature engineering. This is the first part of any model training process.





But what if you had a centralized repository where every time you run a component, you store the result or the update or any other output of that stage into a repository. So whenever any changes made, which causes a different result, you don't need to worry about the progress you've made so far getting lost. You can examine your previous results to try to understand what happened and make corrections or take advantage of improvements. Let's take a closer look. In addition to the executor where your code runs, each component also includes two additional parts, the driver and publisher. The executor is where the work of the component is done and that's what makes different components different. Whatever input is needed for the executor, is provided by the driver, which gets it from the metadata store. Finally, the publisher will push the results of running the executor back into the metadata store. Most of the time, you won't need to customize the driver or publisher. Creating custom components is almost always done by creating a custom executor.

## Metadata: TFX component architecture

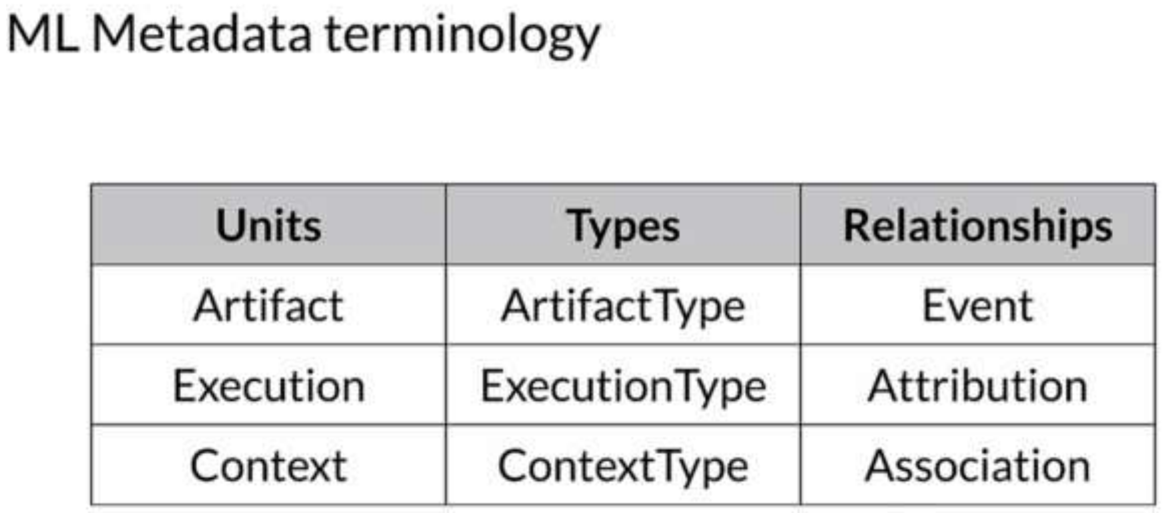


- Driver:
  - Supplies required metadata to executor
- Executor:
  - Place to code the functionality of component
- Publisher:
  - Stores result into metadata

## ML Metadata library

- Tracks metadata flowing between components in pipeline
- Supports multiple storage backends

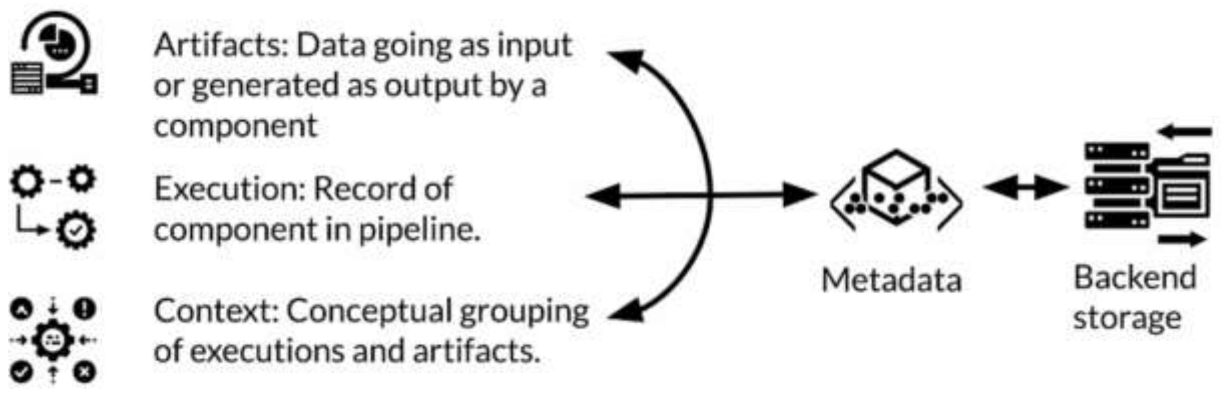
- An artifact is an elementary unit of data that gets fed into the ML metadata store and as the data is consumed as input or generated as output of each component. Next there are executions.
- Each execution is a record of any component run during the ML pipeline workflow, along with its associated runtime parameters. Any artifact or execution will be associated with only one type of component. Artifacts and executions can be clustered together for each type of component separately.
- This grouping is referred to as the context. A context may hold the metadata of the projects being run, experiments being conducted, details about pipelines, etc.



## ML Metadata terminology

| Units | Types | Relationships |
|---|---|---|
| Artifact | ArtifactType | Event |
| Execution | ExecutionType | Attribution |
| Context | ContextType | Association |

## Metadata stored

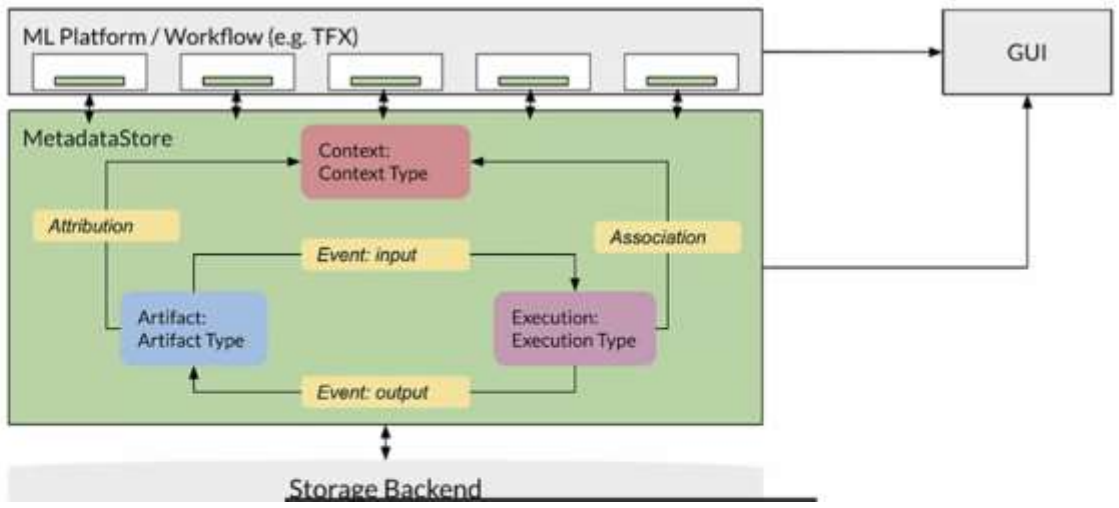


Artifacts: Data going as input or generated as output by a component

Execution: Record of component in pipeline.

Context: Conceptual grouping of executions and artifacts.

## Inside MetadataStore



Relationships store the various units getting generated or consumed when interacting with other units. For example, an event is the record of a relationship between an artifact and an execution. So, ML metadata stores a wide range of information about the results of the components and execution runs of a pipeline. It stores artifacts and it stores the executions of each component in the pipeline. It also stores the lineage information for each artifact that is generated. All of this information is represented in metadata objects and this metadata is stored in a back end storage solution.

## Key points

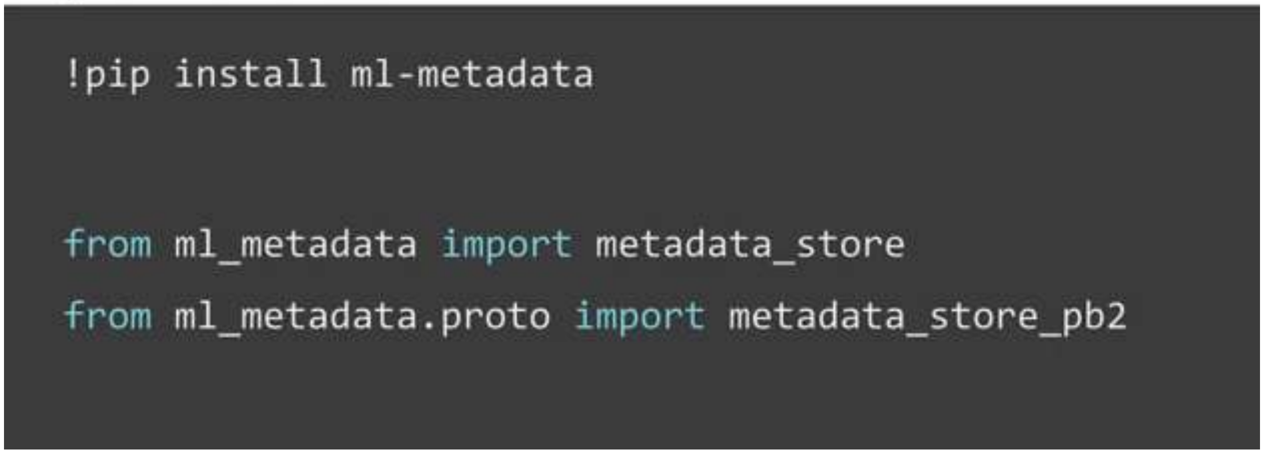
ML metadata:

- Architecture and nomenclature
- Tracking metadata flowing between components in pipeline

## Other benefits of ML Metadata

- the ability to construct a directed acyclic graph or DAG, of the component executions occurring in a pipeline, which can be useful for debugging purposes.
- an verify which inputs have been used in an execution. You can also summarize all the artifacts belonging to a specific type generated after a series of experiments. For example, you can list all of the models that have been trained. You can then compare them to evaluate your various training runs.

## Import ML Metadata



```
!pip install ml-metadata


from ml_metadata import metadata_store
from ml_metadata.proto import metadata_store_pb2
```

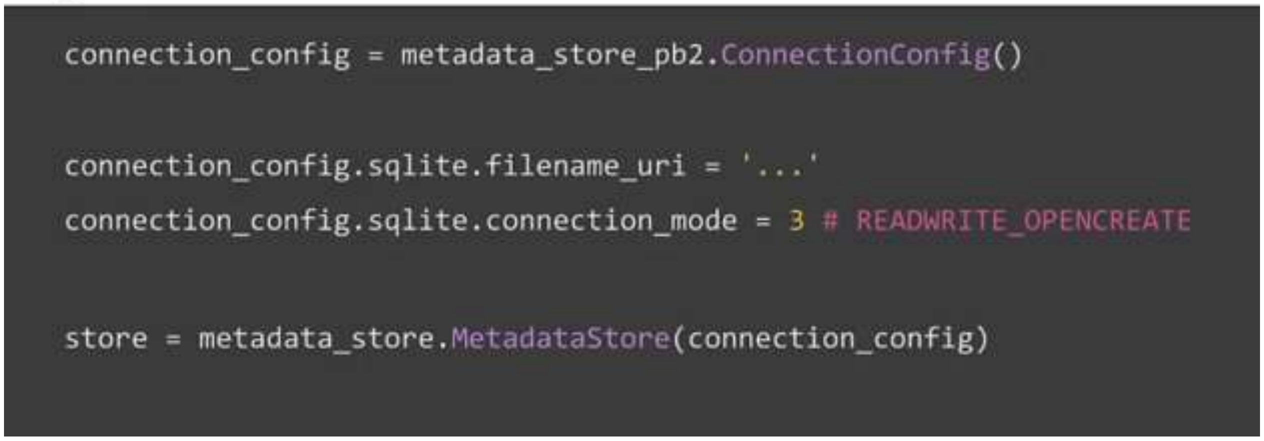
## ML Metadata storage backend

- ML metadata registers metadata in a database called Metadata Store
- APIs to record and retrieve metadata to and from the storage backend:
    - Fake database: in-memory for fast experimentation/prototyping
    - SQLite: in-memory and disk
    - MySQL: server based
    - Block storage: File system, storage area network, or cloud based

## Fake database

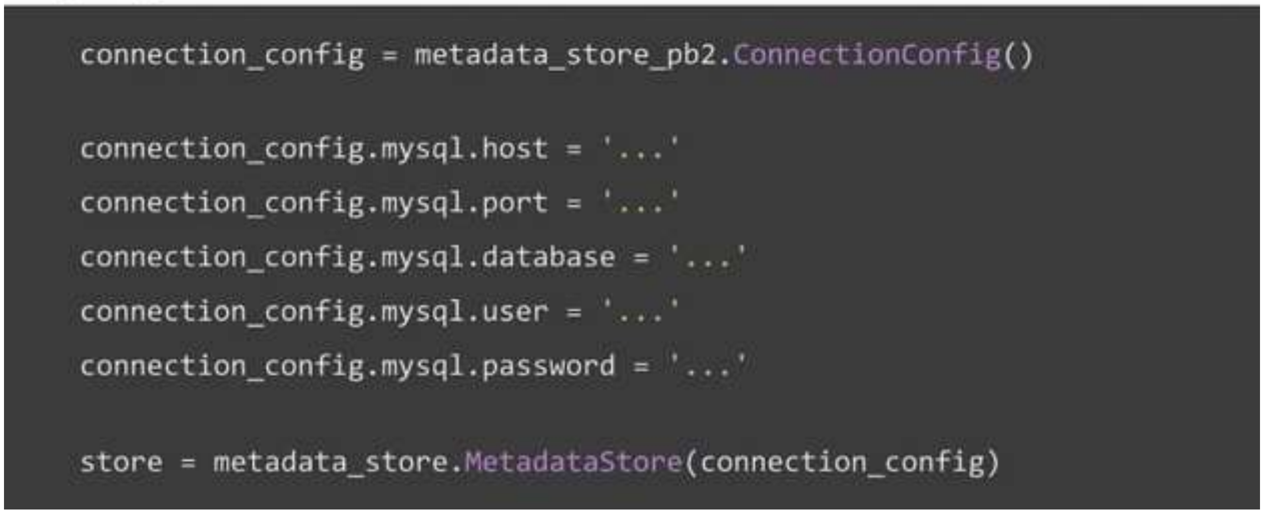
```
connection_config = metadata_store_pb2.ConnectionConfig()

# Set an empty fake database proto
connection_config.fake_database.SetInParent()



store = metadata_store.MetadataStore(connection_config)
```

## SQLite



```
connection_config = metadata_store_pb2.ConnectionConfig()

connection_config.sqlite.filename_uri = '...'
connection_config.sqlite.connection_mode = 3 # READWRITE_OPENCREATE

store = metadata_store.MetadataStore(connection_config)
```

## MySQL



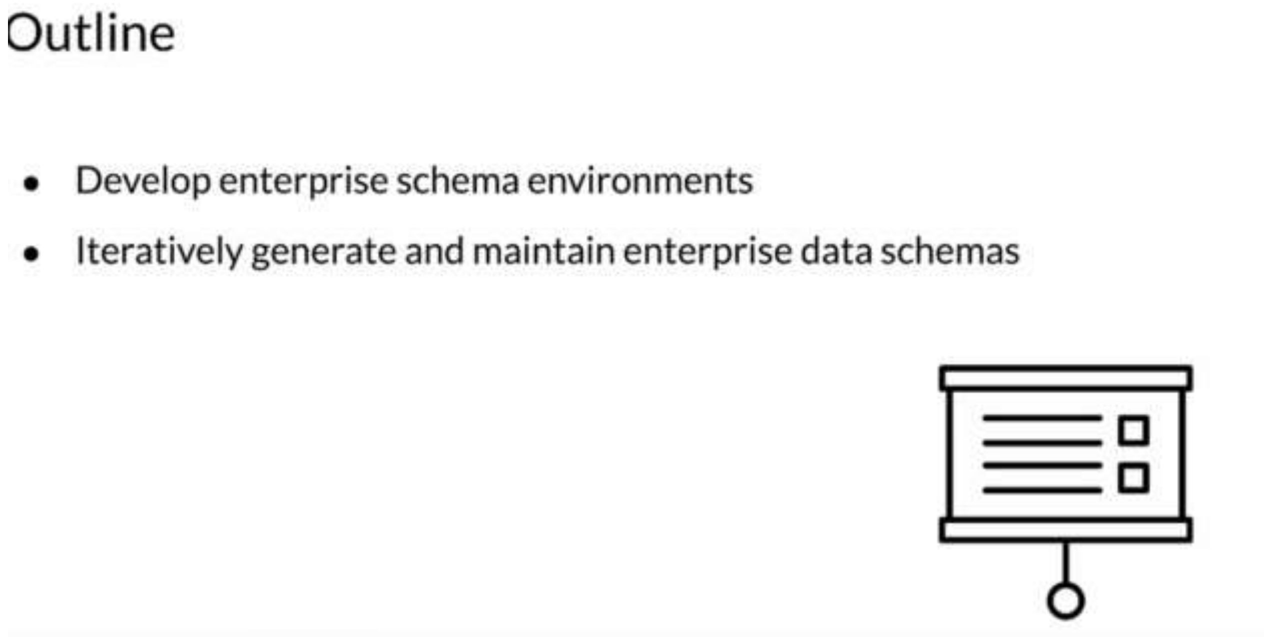
```
connection_config = metadata_store_pb2.ConnectionConfig()

connection_config.mysql.host = '...'
connection_config.mysql.port = '...'
connection_config.mysql.database = '...'
connection_config.mysql.user = '...'
connection_config.mysql.password = '...'

store = metadata_store.MetadataStore(connection_config)
```

## ML metadata practice: ungraded lab

- Using a tabular data set, you will explore:
    - Explicit programming in ML Metadata
    - Integration with TFDV
    - Store progress and create provisions to backtrack the experiment

## Key points

- Walk through over the data journey addressing lineage and provenance
- The importance of metadata for tracking data evolution
- ML Metadata library and its usefulness to track data changes
- Running an example to register artifacts, executions, and contexts

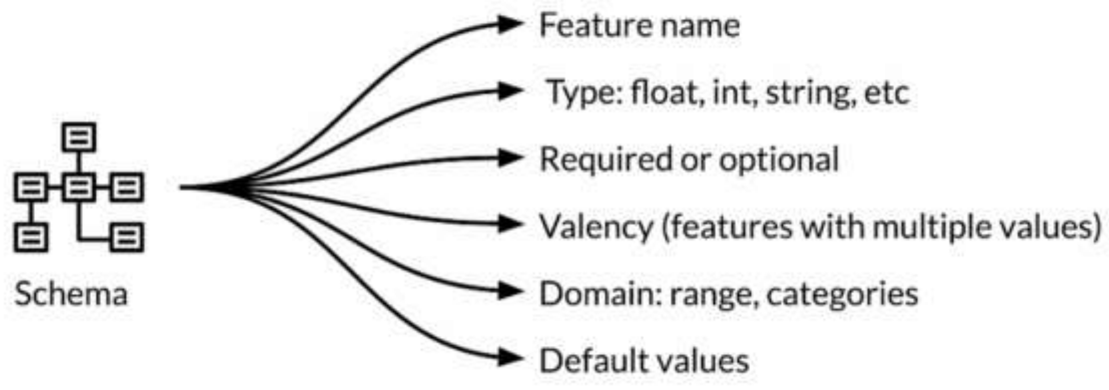


## Outline

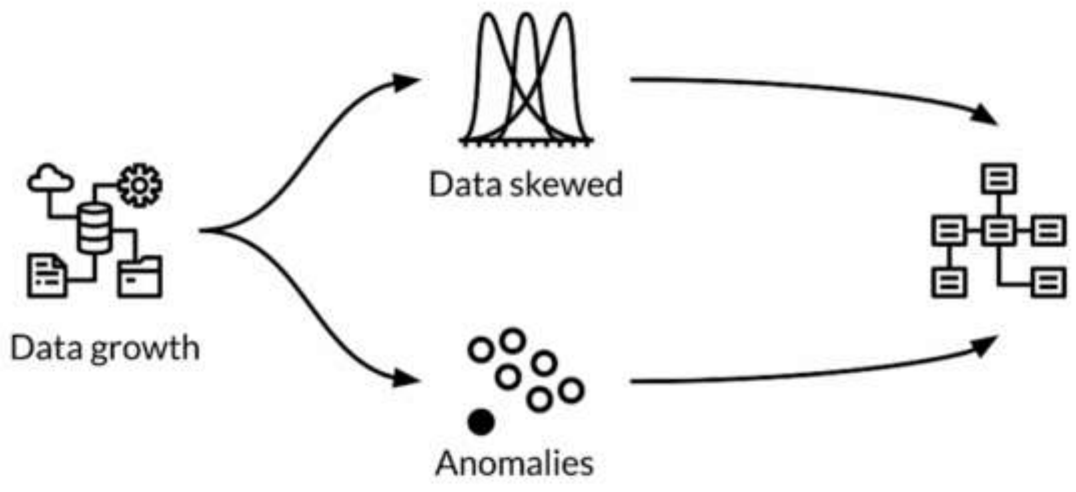
- Develop enterprise schema environments
- Iteratively generate and maintain enterprise data schemas

Schemas are relational objects summarizing the features in a given dataset or project. They include the feature name, the feature of variable type. For example, an integer, float, string or categorical variable. Whether or not the feature is required. The valency of the feature, which applies to features with multiple values like lists or array features, and expresses the minimum and maximum number of values. Information about the range and categories and feature default values. Schemas are important, as your data and feature set evolves over time. From your experience, you know that data keeps changing and this change often results in change distributions.

## Review: Recall Schema


Schema
Feature name
Type: float, int, string, etc
Required or optional
Valency (features with multiple values)
Domain: range, categories
Default values

## Iterative schema development & evolution



## Reliability during data evolution

Platform needs to be resilient to disruptions from:

- Inconsistent data
- Software
- User configurations
- Execution environments

## Scalability during data evolution

Platform must scale during:

- High data volume during training
- Variable request traffic during serving

## Anomaly detection during data evolution

Platform designed with these principles:

Easy to detect anomalies

Data errors treated same as code bugs

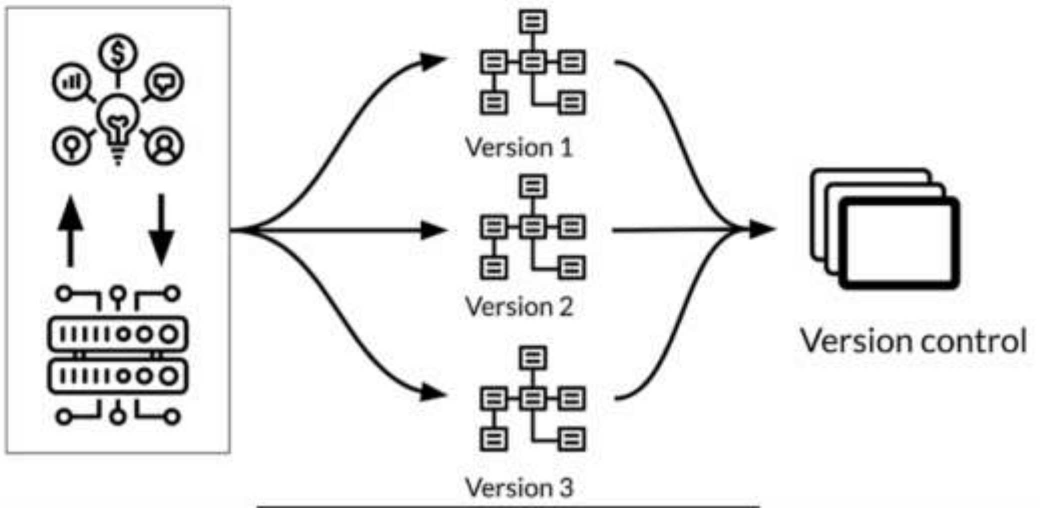
Update data schema

## Schema inspection during data evolution

Looking at schema versions to track data evolution

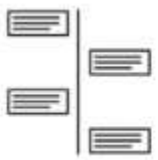
Schema can drive other automated processes

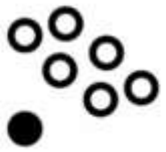
## Multiple schema versions



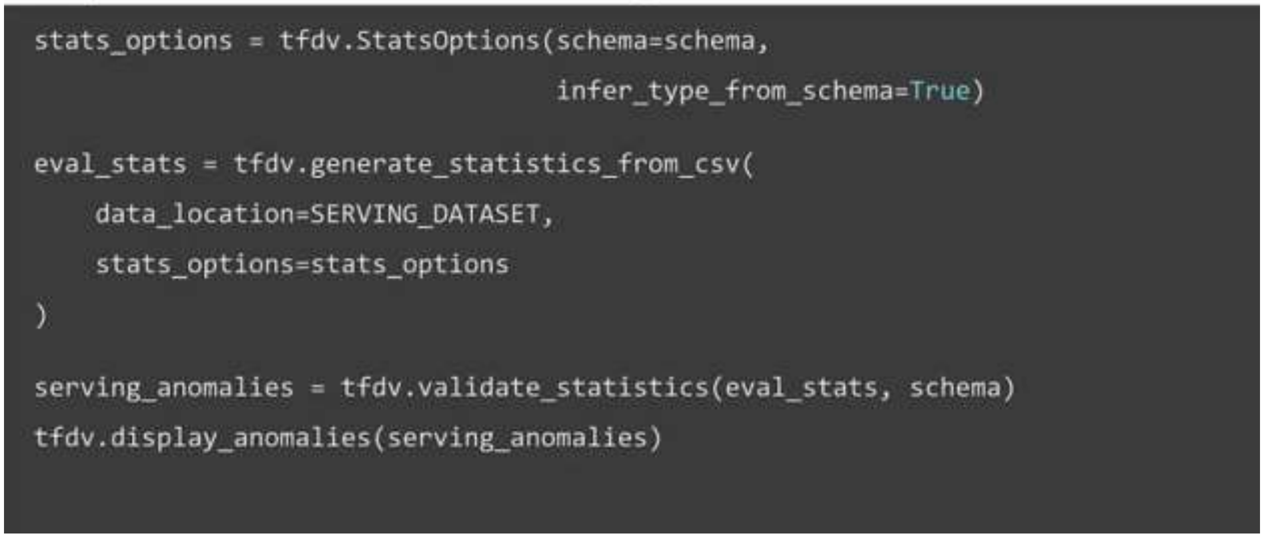
## Maintaining varieties of schema

Business use-case needs to support data from different sources.


Data evolves rapidly

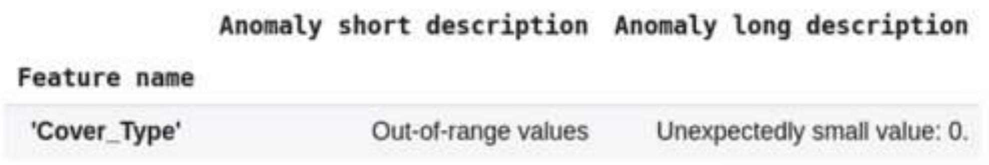

Is anomaly part of accepted type of data?

## Inspect anomalies in serving dataset



```
stats_options = tfdv.StatsOptions(schema=schema,
                                  infer_type_from_schema=True)

eval_stats = tfdv.generate_statistics_from_csv(
    data_location=SERVING_DATASET,
    stats_options=stats_options
)

serving_anomalies = tfdv.validate_statistics(eval_stats, schema)
tfdv.display_anomalies(serving_anomalies)
```

## Anomaly: No labels in serving dataset



| Feature name | Anomaly short description | Anomaly long description |
|---|---|---|
| **'Cover_Type'** | Out-of-range values | Unexpectedly small value: 0. |

## Schema environments

- Customize the schema for each environment
- Ex: Add or remove label in schema based on type of dataset

## Create environments for each schema

```
schema.default_environment.append('TRAINING')
schema.default_environment.append('SERVING')

tfdv.get_feature(schema, 'Cover_Type')
    .not_in_environment.append('SERVING')
```

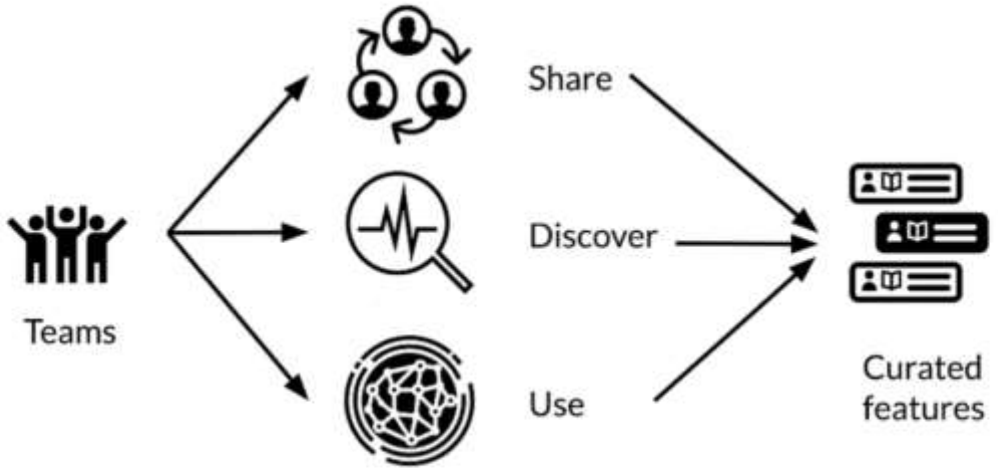
## Inspect anomalies in serving dataset

```
serving_anomalies = tfdv.validate_statistics(eval_stats,
                                             schema,
                                             environment='SERVING')


tfdv.display_anomalies(serving_anomalies)
# No anomalies found
```

## Key points

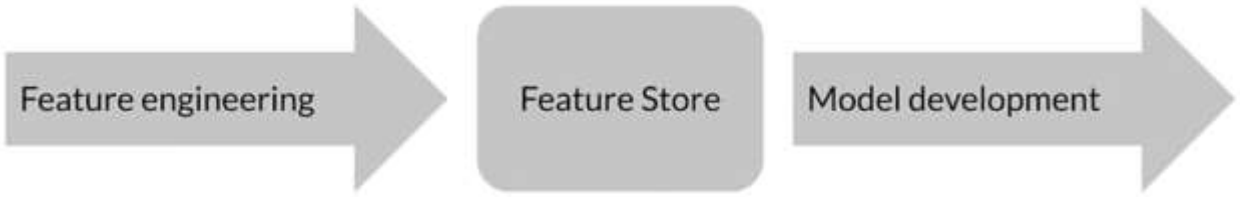
- Iteratively update and fine-tune schema to adapt to evolving data
- How to deal with scalability and anomalies
- Set schema environments to detect anomalies in serving requests

## Feature stores

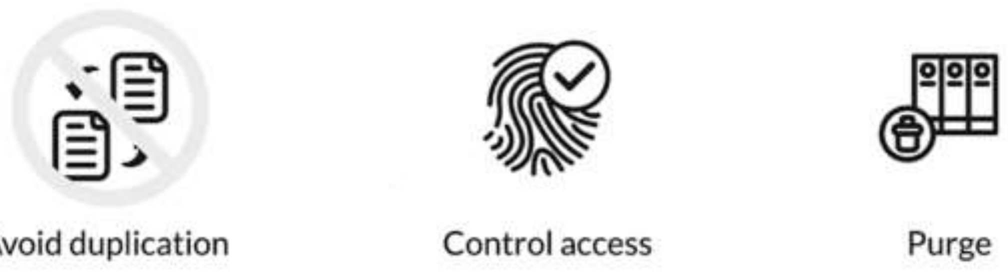


## Feature stores

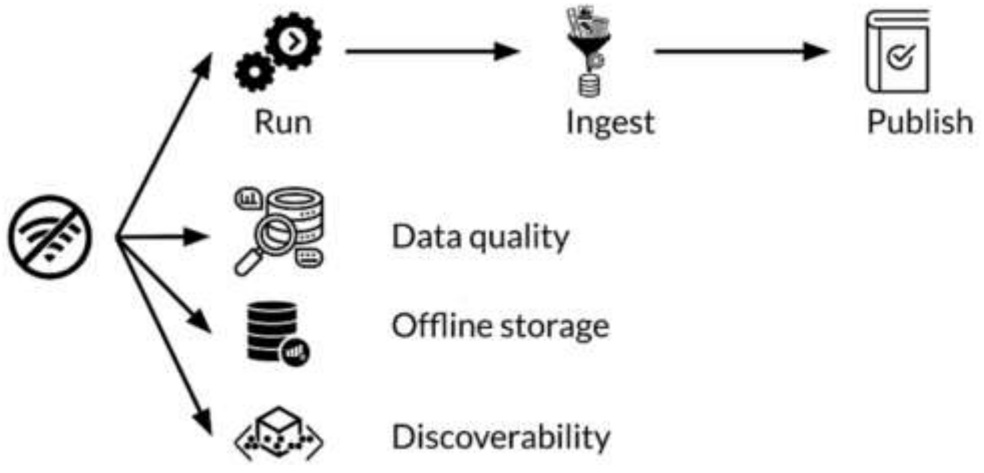
Many modeling problems use identical or similar features

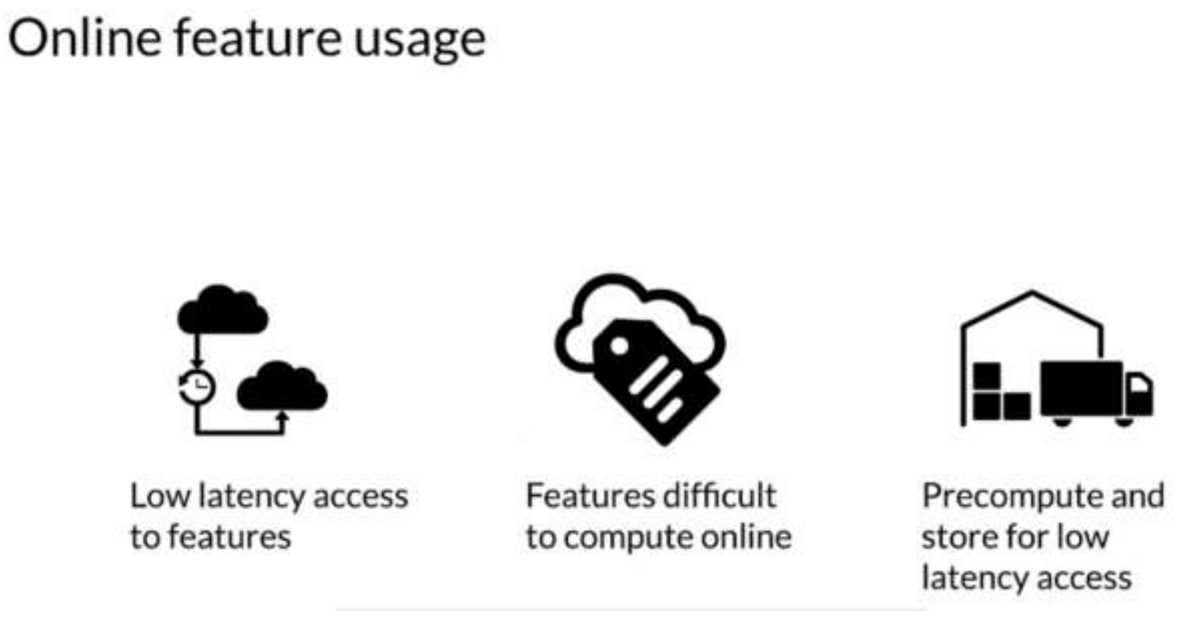


## Feature stores



## Offline feature processing



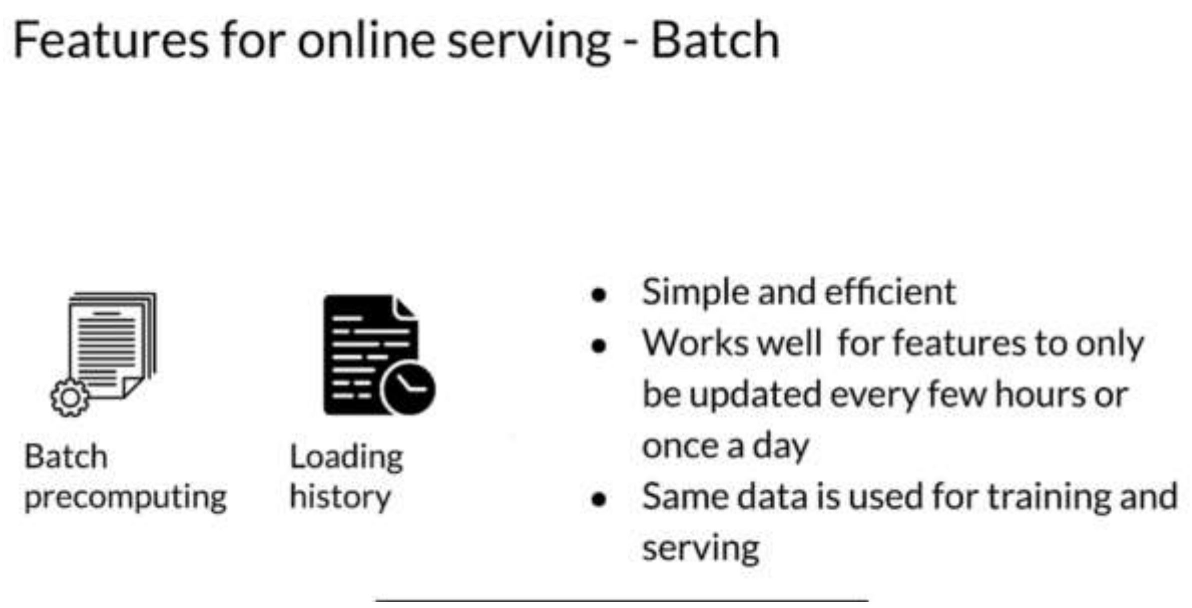


## Online feature usage

Low latency access to features

Features difficult to compute online

Precompute and store for low latency access



## Features for online serving - Batch

Batch precomputing

Loading history

- Simple and efficient
- Works well  for features to only be updated every few hours or once a day
- Same data is used for training and serving

## Feature store: key aspects

- Managing feature data from a single person to large enterprises.
- Scalable and performant access to feature data in training and serving.
- Provide consistent and point-in-time correct access to feature data.
- Enable discovery, documentation, and insights into your features.

## Data warehouse

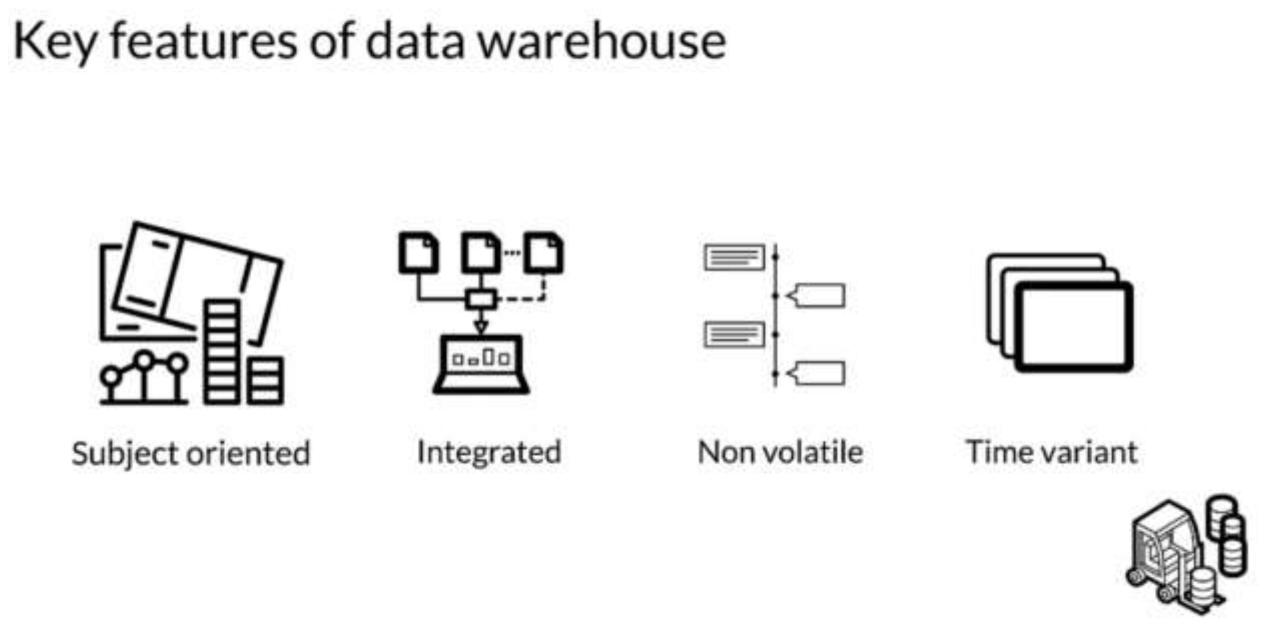
Aggregates data sources

Processed and analyzed

Read optimized

Not real time

Follows schema



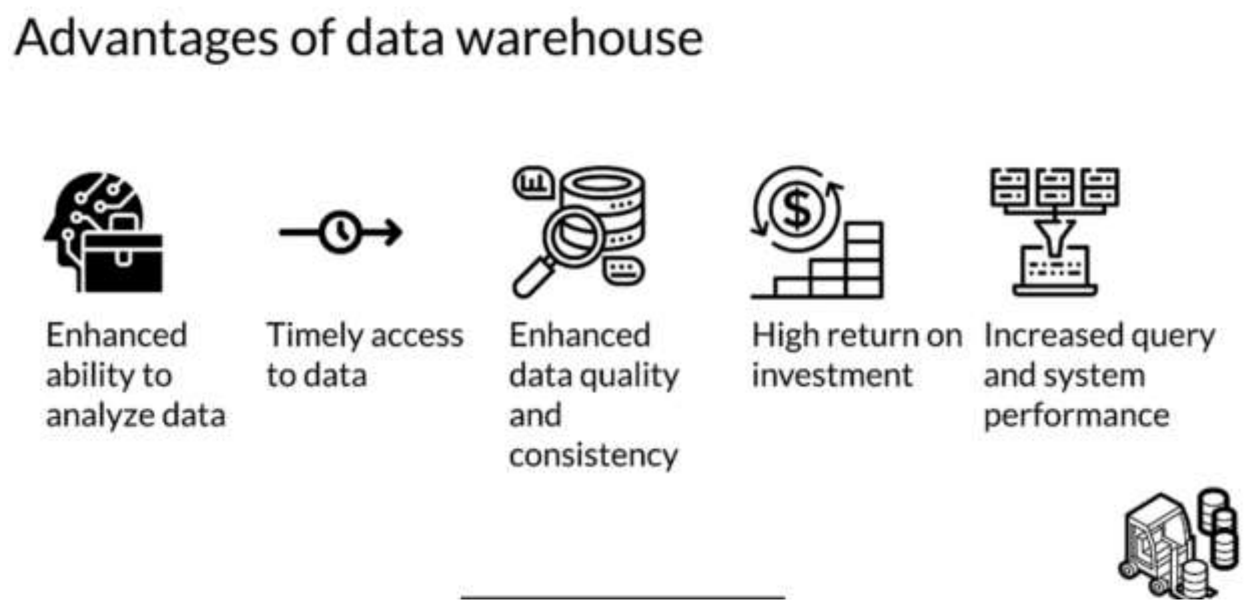
## Key features of data warehouse

Subject oriented

Integrated

Non volatile

Time variant



## Advantages of data warehouse

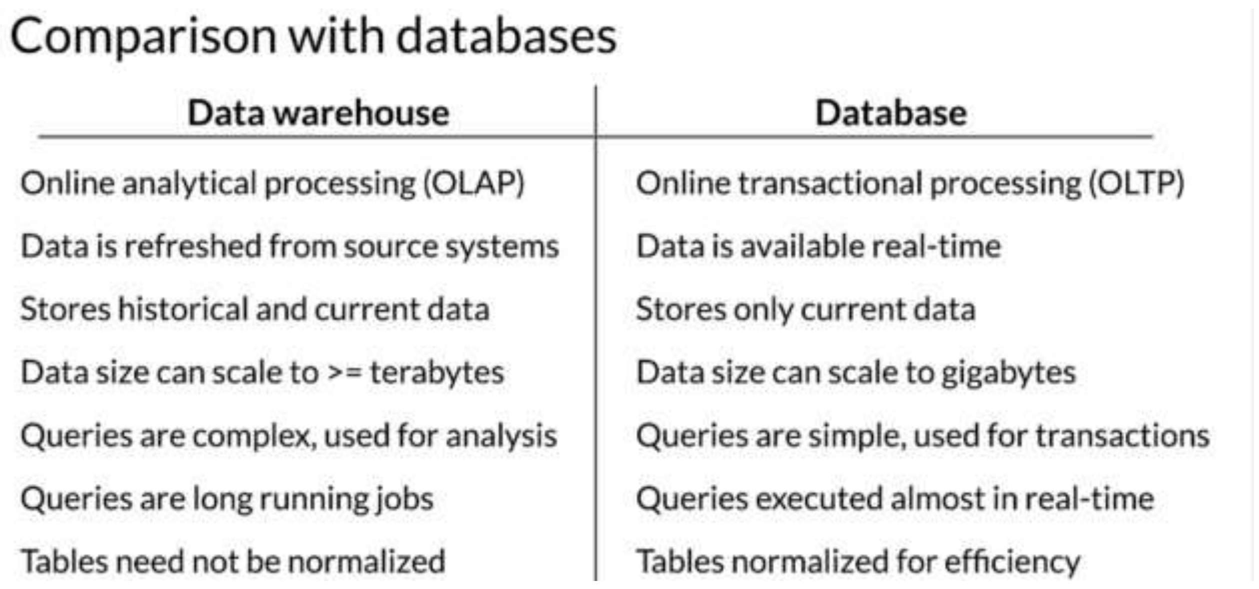
Enhanced ability to analyze data

Timely access to data

Enhanced data quality and consistency

High return on investment

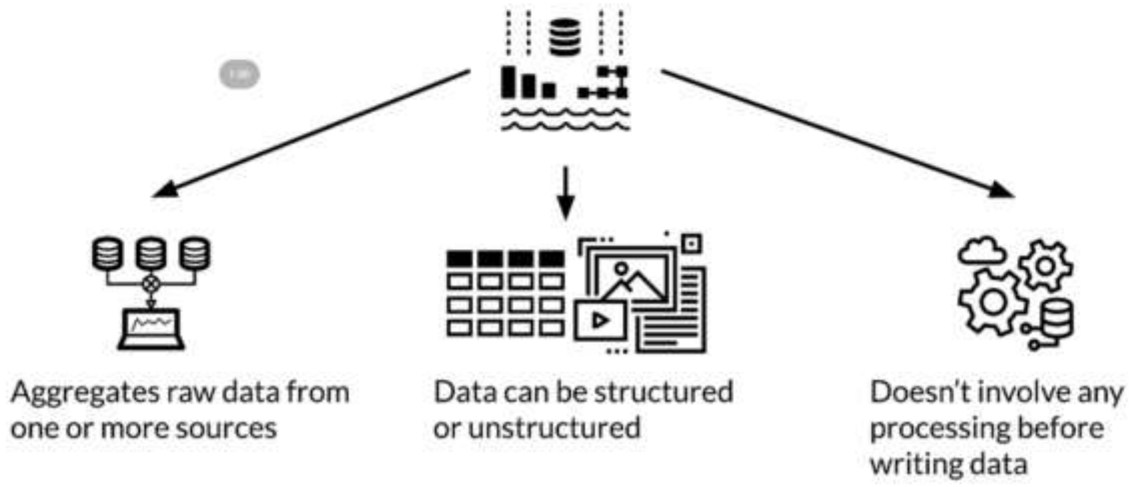
Increased query and system performance



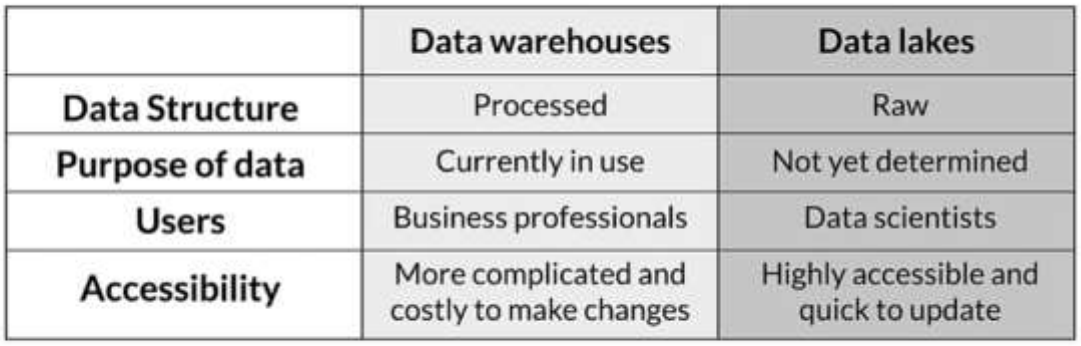
## Comparison with databases

| Data warehouse | Database |
|---|---|
| Online analytical processing (OLAP) | Online transactional processing (OLTP) |
| Data is refreshed from source systems | Data is available real-time |
| Stores historical and current data | Stores only current data |
| Data size can scale to >= terabytes | Data size can scale to gigabytes |
| Queries are complex, used for analysis | Queries are simple, used for transactions |
| Queries are long running jobs | Queries executed almost in real-time |
| Tables need not be normalized | Tables normalized for efficiency |

## Data lakes


Aggregates raw data from one or more sources
Data can be structured or unstructured
Doesn't involve any processing before writing data

## Comparison with data warehouse



| | Data warehouses | Data lakes |
|---|---|---|
| **Data Structure** | Processed | Raw |
| **Purpose of data** | Currently in use | Not yet determined |
| **Users** | Business professionals | Data scientists |
| **Accessibility** | More complicated and costly to make changes | Highly accessible and quick to update |

## Key points

- **Feature store**: central repository for storing documented, curated, and access-controlled features, specifically for ML.
- **Data warehouse**: subject-oriented repository of structured data optimized for fast read.
- **Data lakes**: repository of data stored in its natural and raw format.