Data Lifecycle in Production

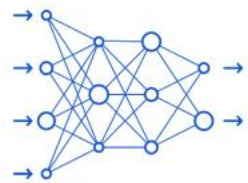
Part 3: Data Journey

Ramin Toosi

The data journey

Raw features and labels

Input-output map



ML model to learn mapping

Data transformation

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

Artifacts and the ML pipeline

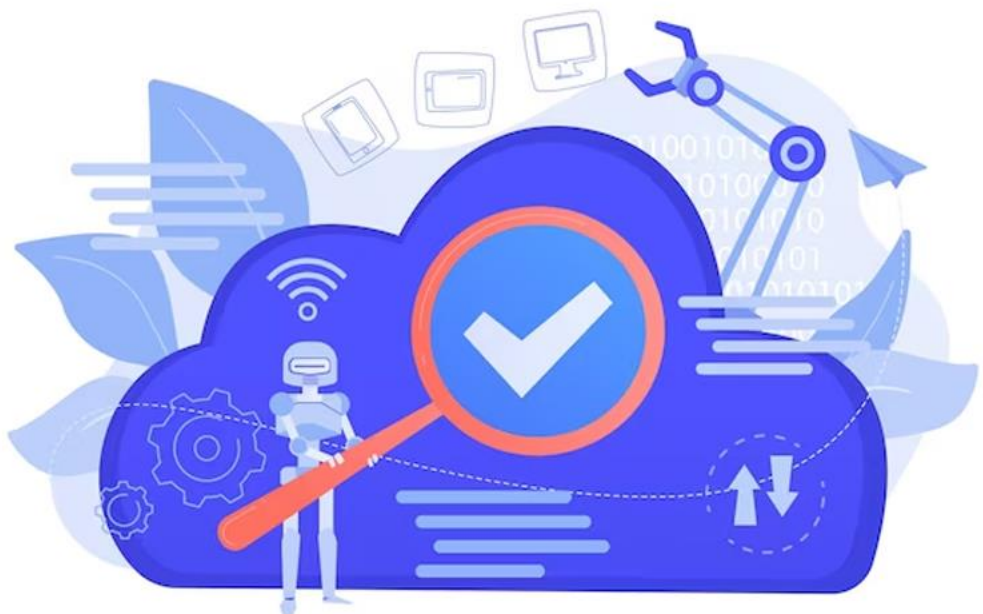
- Artifacts are created as the components of the ML pipeline execute
- Artifacts include all of the data and objects which are produced by the pipeline components
- This includes the data, in different stages of transformation, the schema, the model itself, metrics, etc.

Data provenance and lineage

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

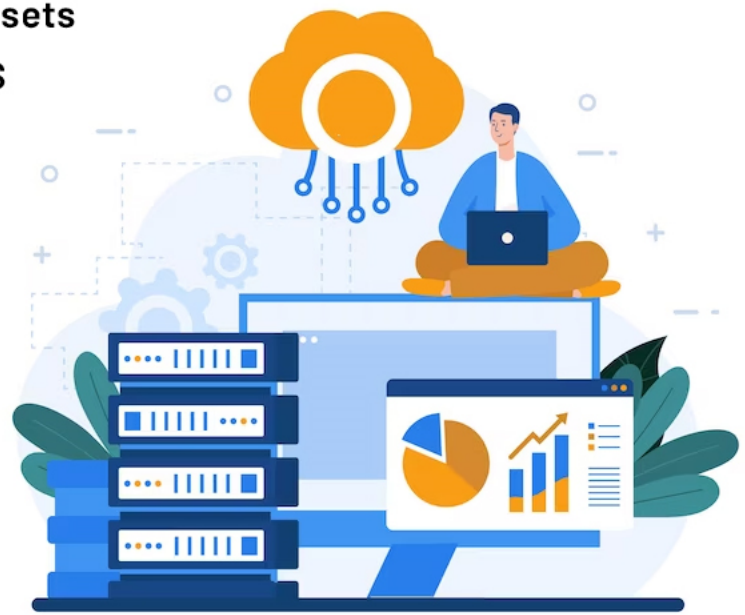
Data lineage: data protection regulation

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

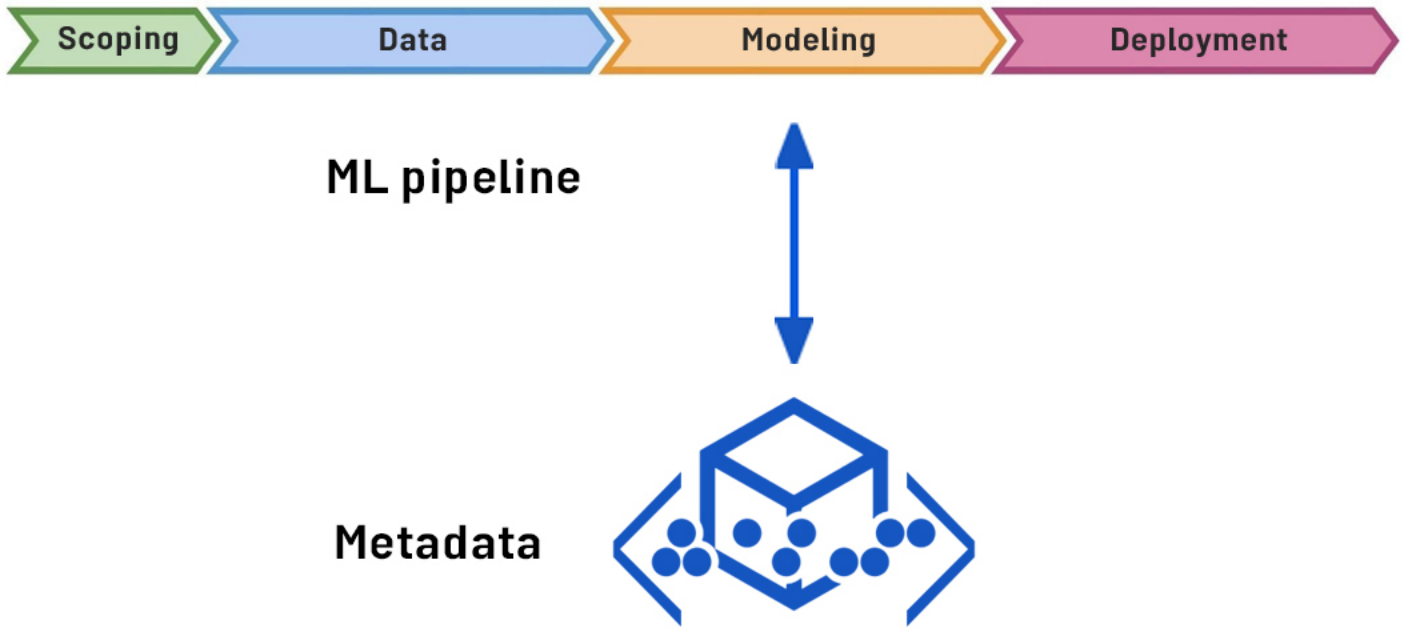


Data versioning

- 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

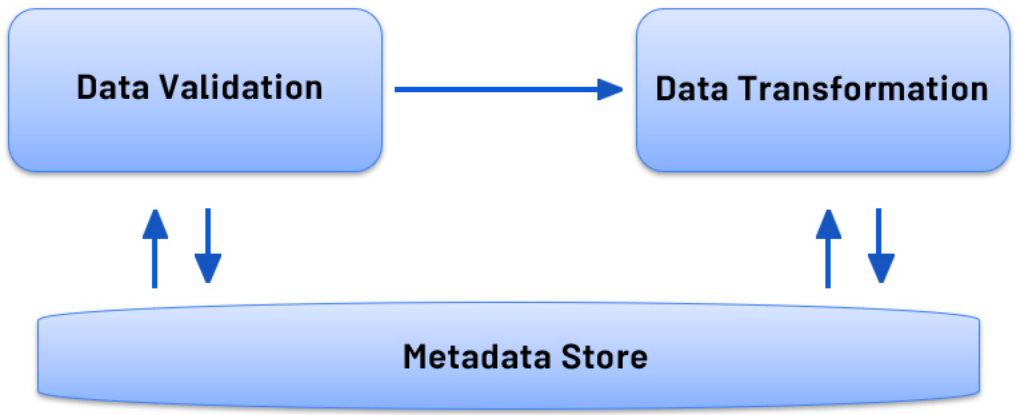


Metadata: Tracking artifacts and pipeline changes



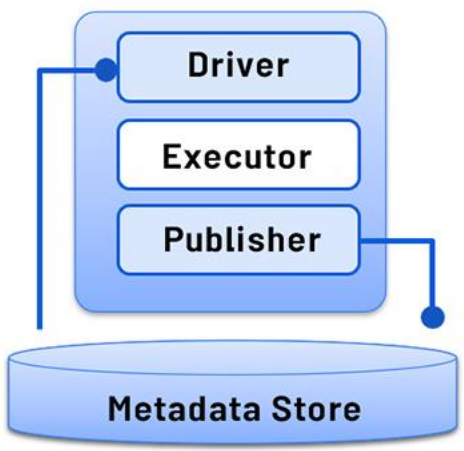
Ordinary ML data pipeline

Metadata: Tracking progress



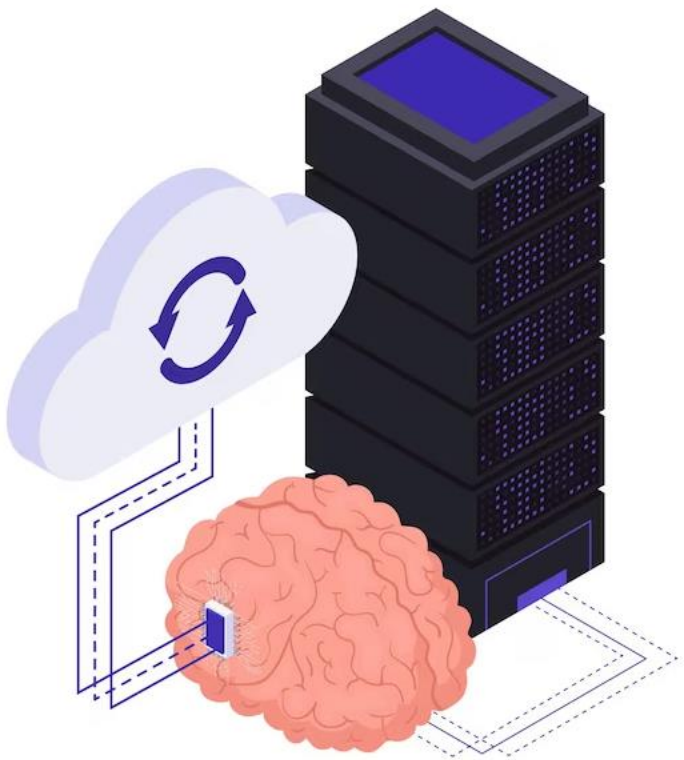
Metadata: TFX component architecture

- **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

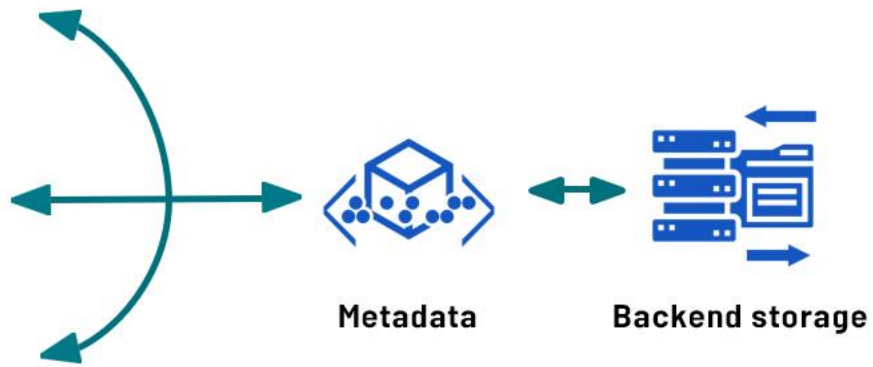


Metadata stored

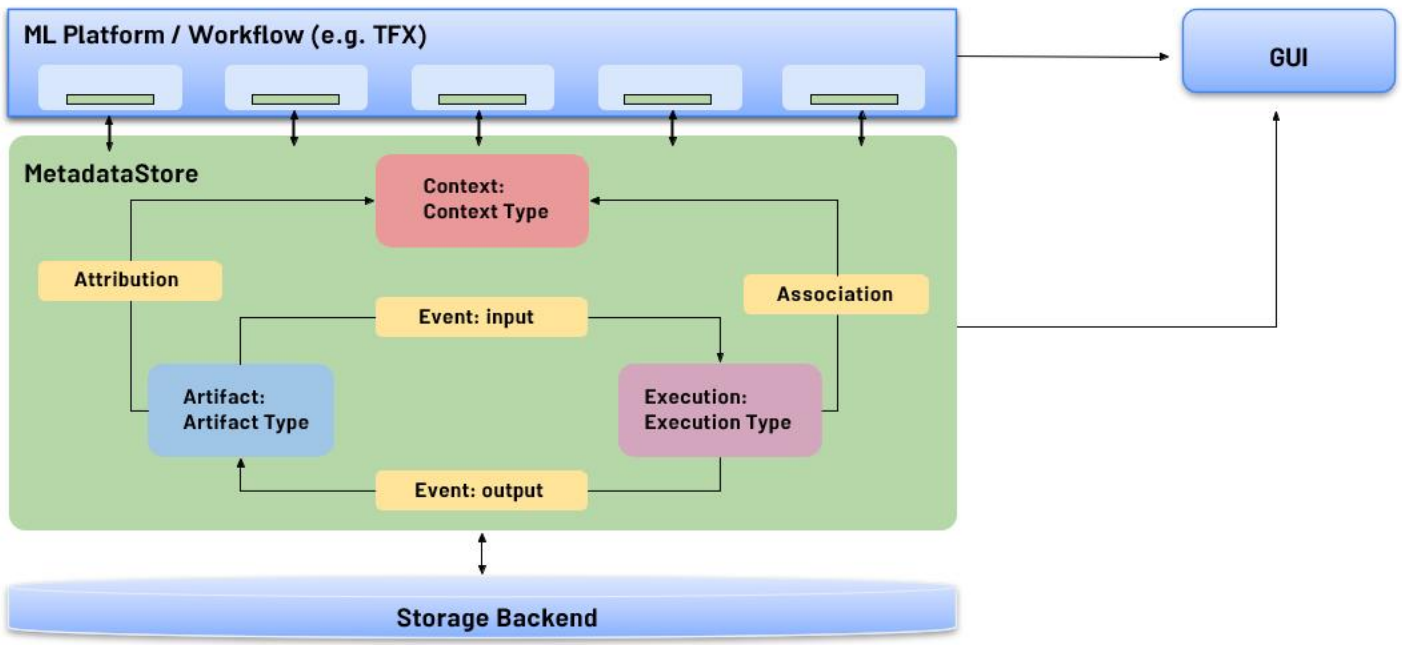
Artifacts: Data going as input or generated as output by a component

Execution: Record of component in pipeline.

Context: Conceptual grouping of executions and artifacts.



Inside MetadataStore



Other benefits of ML Metadata

Produce DAG of pipelines

Verify the inputs used in an execution

List all artifacts

Compare artifacts

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

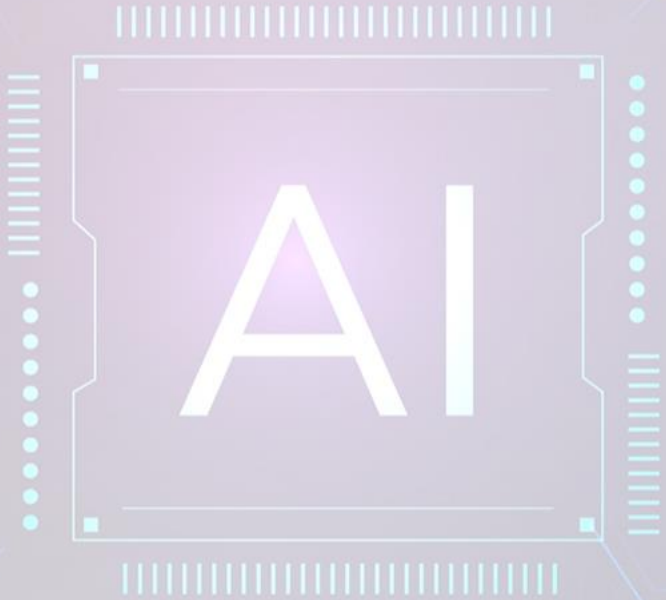
```
connection_config = metadata_store_pb2.ConnectionConfig()

# 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)
```


```
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)
```

Evolving Data



Review: Recall Schema

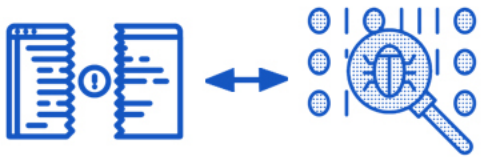
Schema

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

Anomaly detection during data evolution

Platform designed with these principles

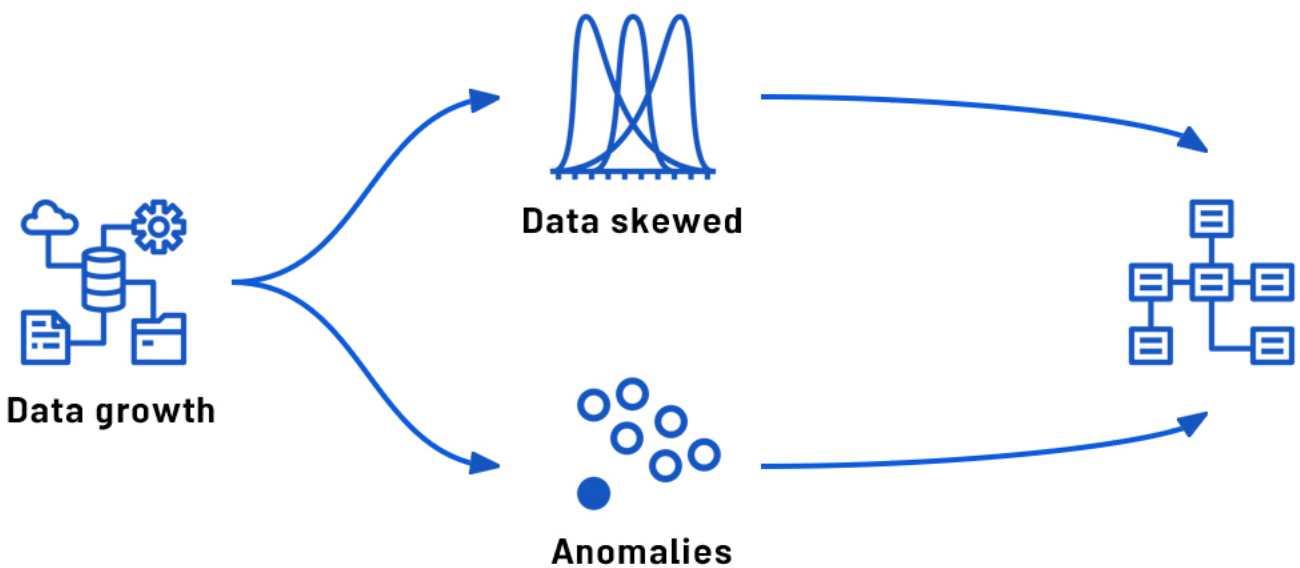
Easy to detect anomalies



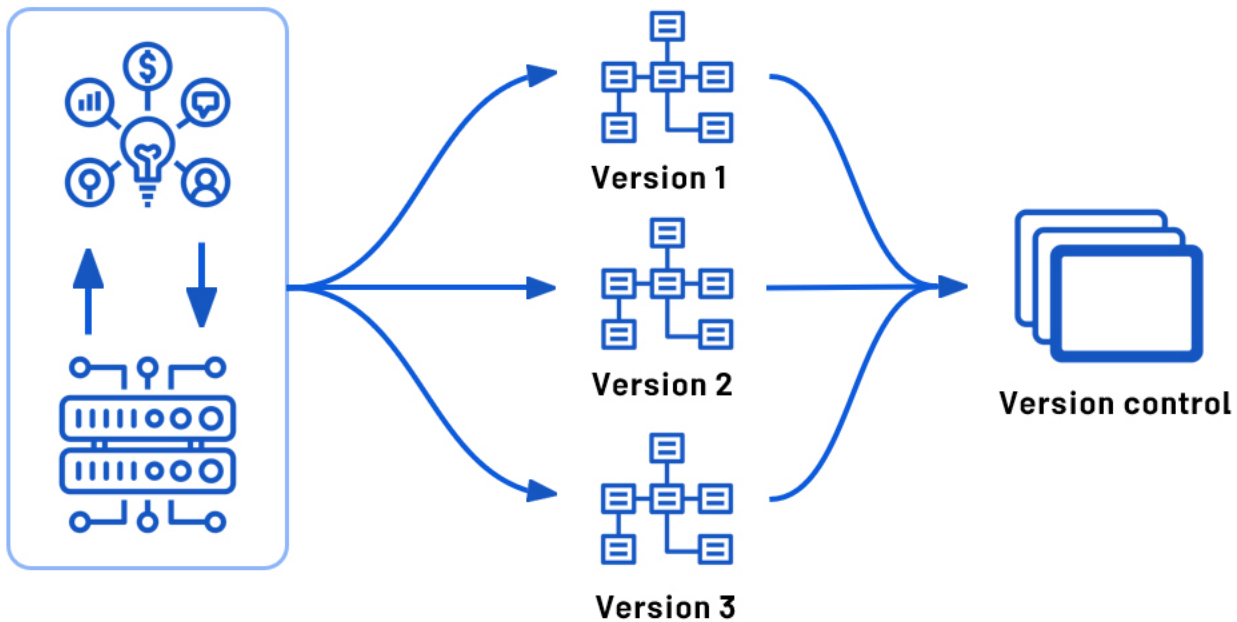
Data errors treated same as code bugs

Update data schema

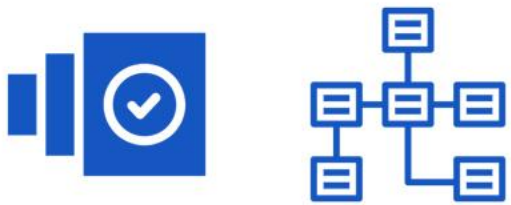
Iterative schema development & evolution



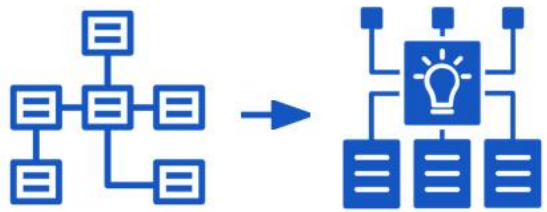
Multiple schema versions



Schema inspection during data evolution



Looking at schema versions to track data evolution



Schema can drive other automated processes

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

Anomaly short description anomaly long description

Feature name		
'Cover_Type'	Out-of-rang 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

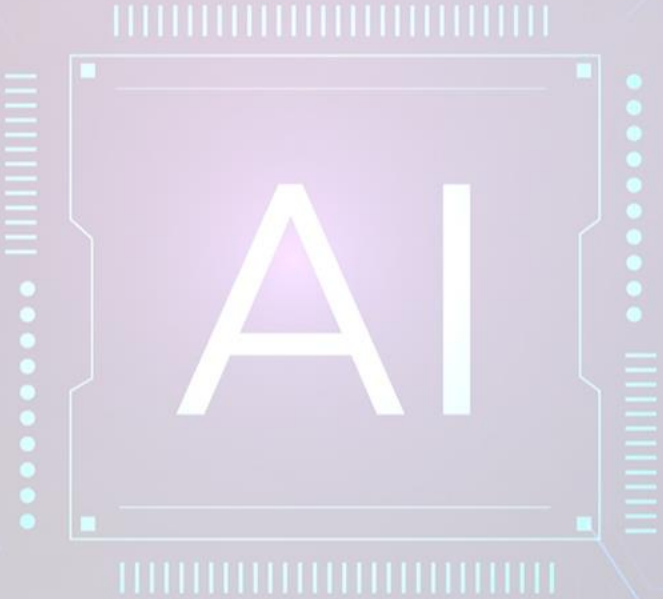
Create environments for each schema

```
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
```

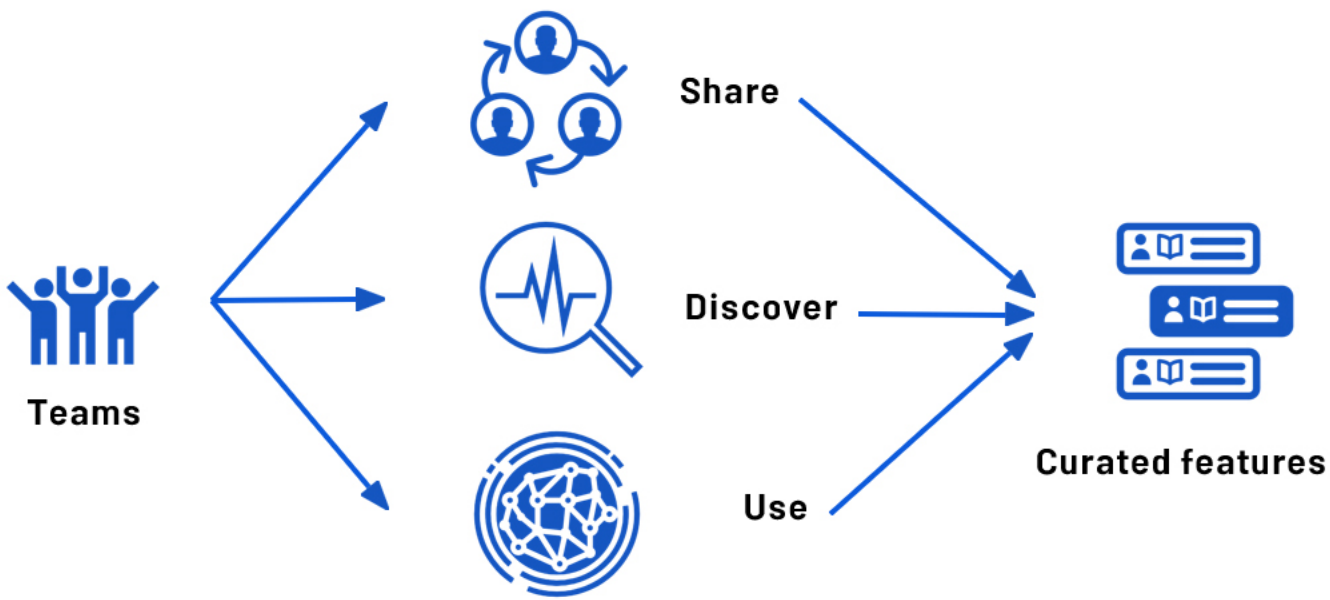
Data Storage



Feature stores

- Many modeling problems use identical or similar features

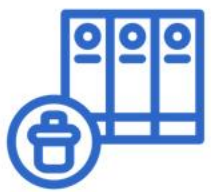
Feature stores



Feature stores

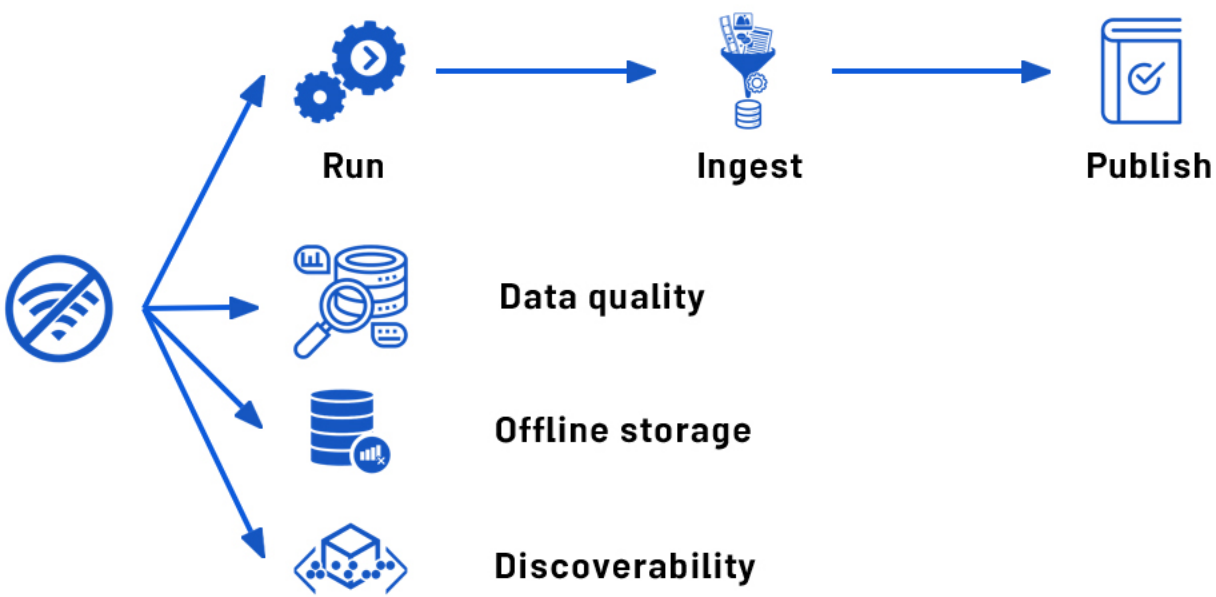
Avoid duplication

Control access



Purge

Offline feature processing



Online feature usage

Low latency access to features

Features difficult to compute online

Precompute and store for low latency access

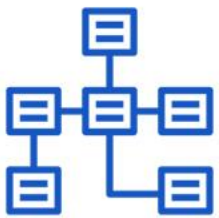
Data warehouse

Aggregates data sources

Processed and analyzed

Read optimized

Not real time

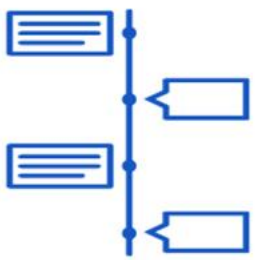


Follows schema

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

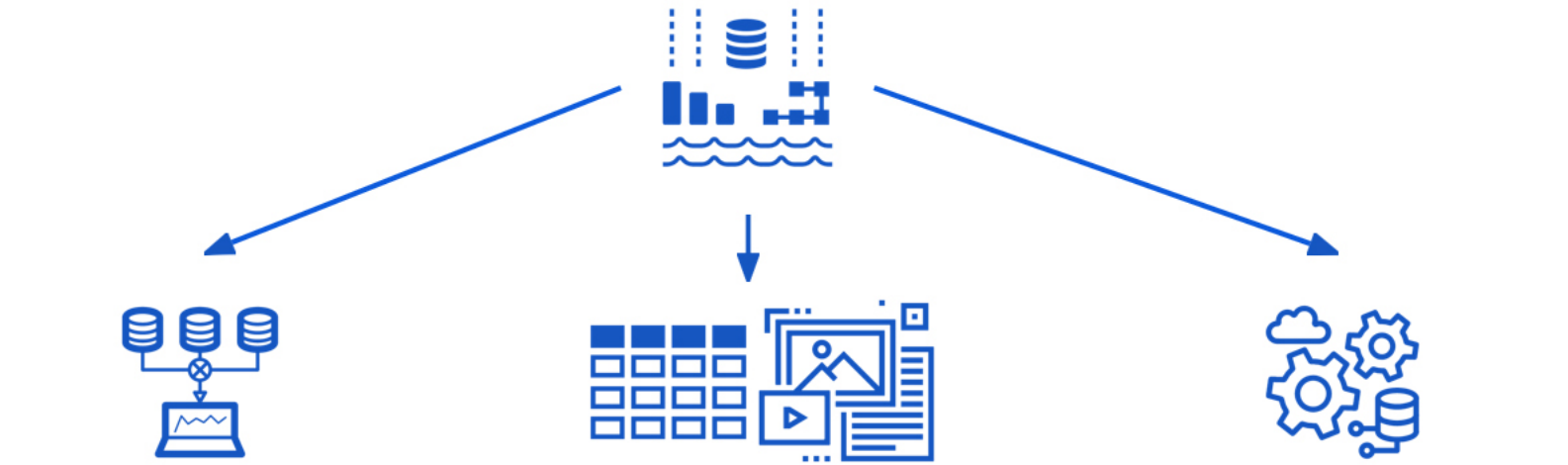
High return on query investment

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 long running jobs	Queries are simple, used for transactions 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

Data lakes vs 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