Copyright Notice

DeepLearning.AI

Data Journey and Data Storage

Welcome

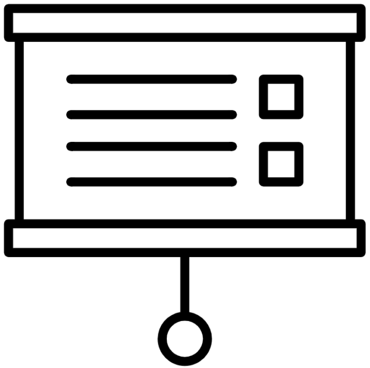
DeepLearning.AI

Data Journey and Data Storage

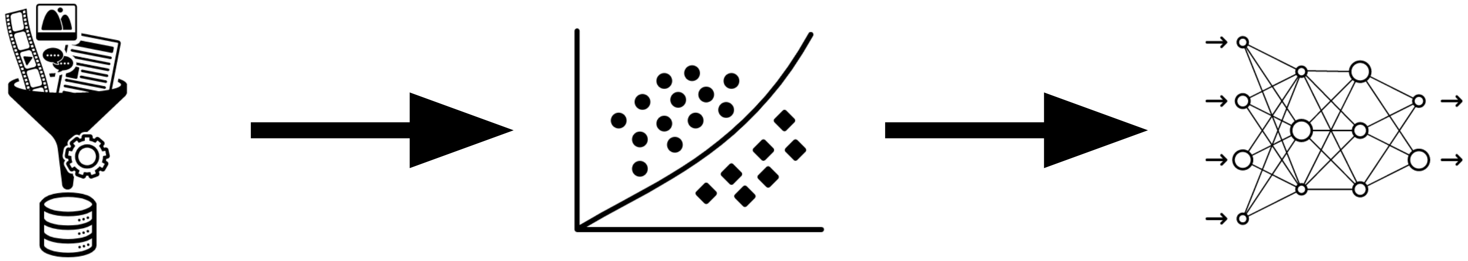
Data Journey

Outline

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



The data journey

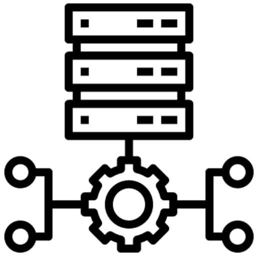


Raw features and labels

Input-output map

ML model to learn mapping

Data transformation



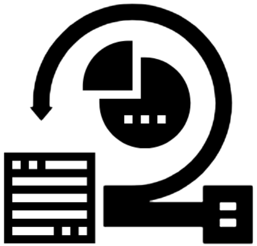
- Data transforms as it flows through the process
- Interpreting model results requires understanding data transformation

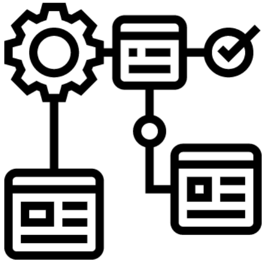
Artifacts and the ML pipeline

- 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

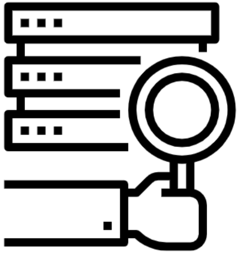
- The chain of transformations that led to the creation of a particular artifact.
- Important for debugging and reproducibility.





Data provenance: Why it matters

Helps with debugging and understanding the ML pipeline:



Inspect artifacts at each point in the training process

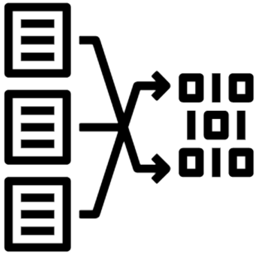
Trace back through a training run

Compare training runs

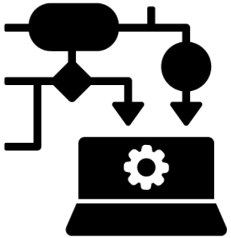
Data lineage: data protection regulation

- 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

DeepLearning.AI

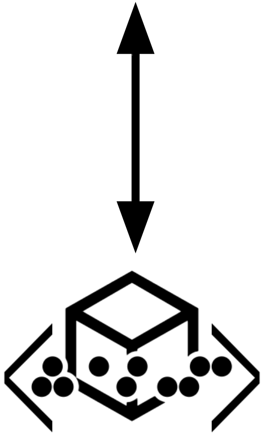
Data Journey and Data Storage

Intro to ML Metadata

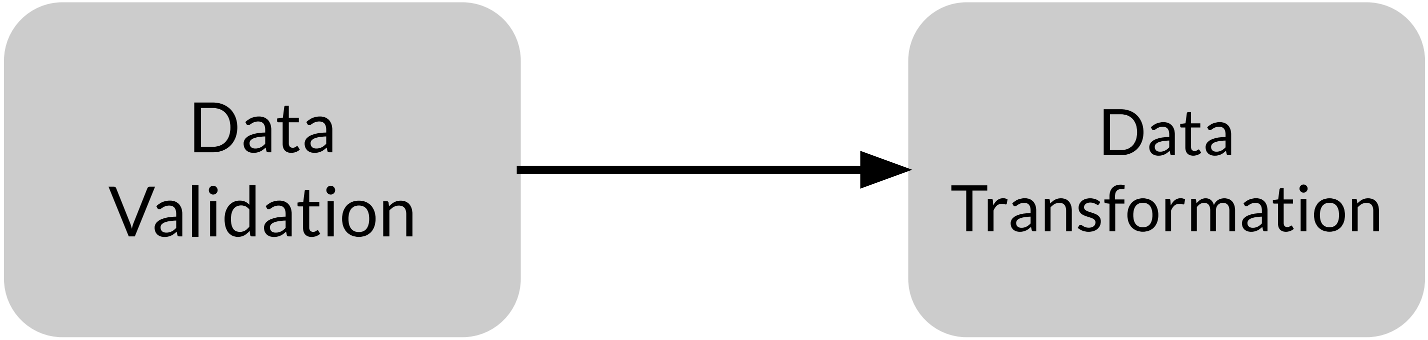
Metadata: Tracking artifacts and pipeline changes

ML pipeline

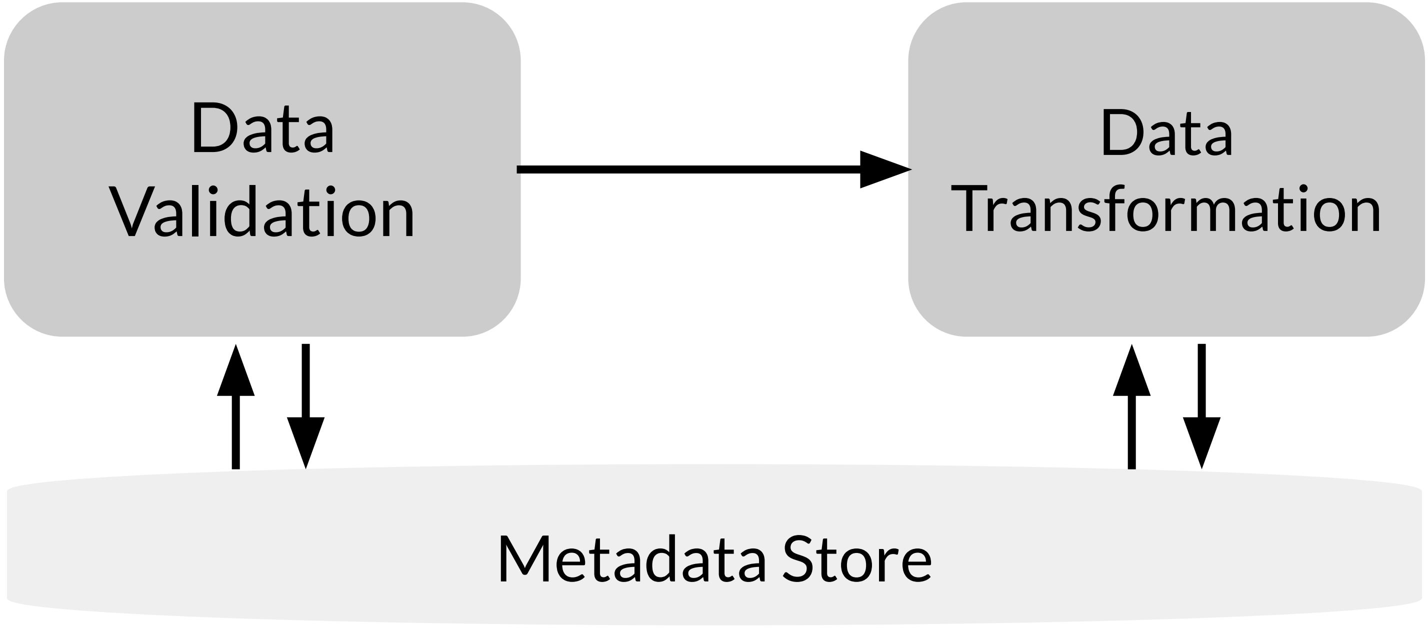
Metadata



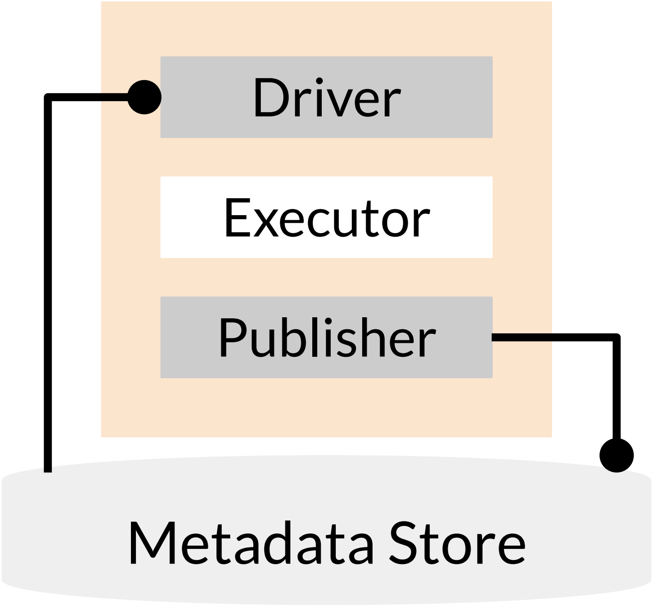
Ordinary ML data pipeline



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

ML Metadata terminology

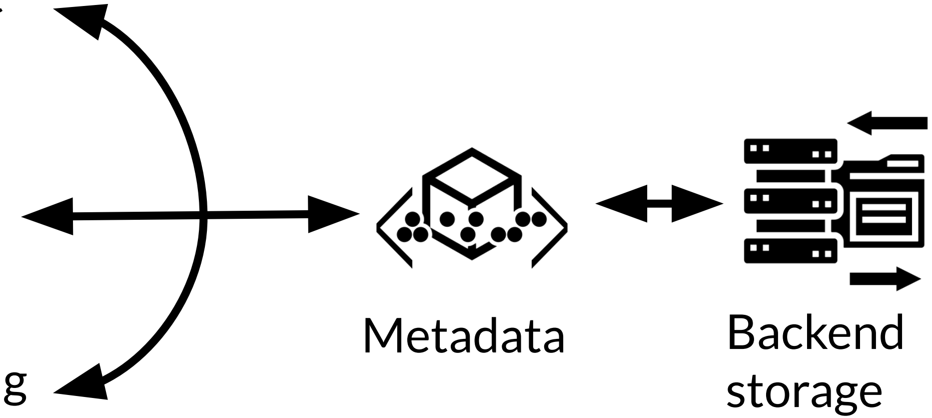
Units	Types	Relationships
Artifact	ArtifactType	Event
Execution	ExecutionType	Attribution
Context	ContextType	Association

Metadata stored

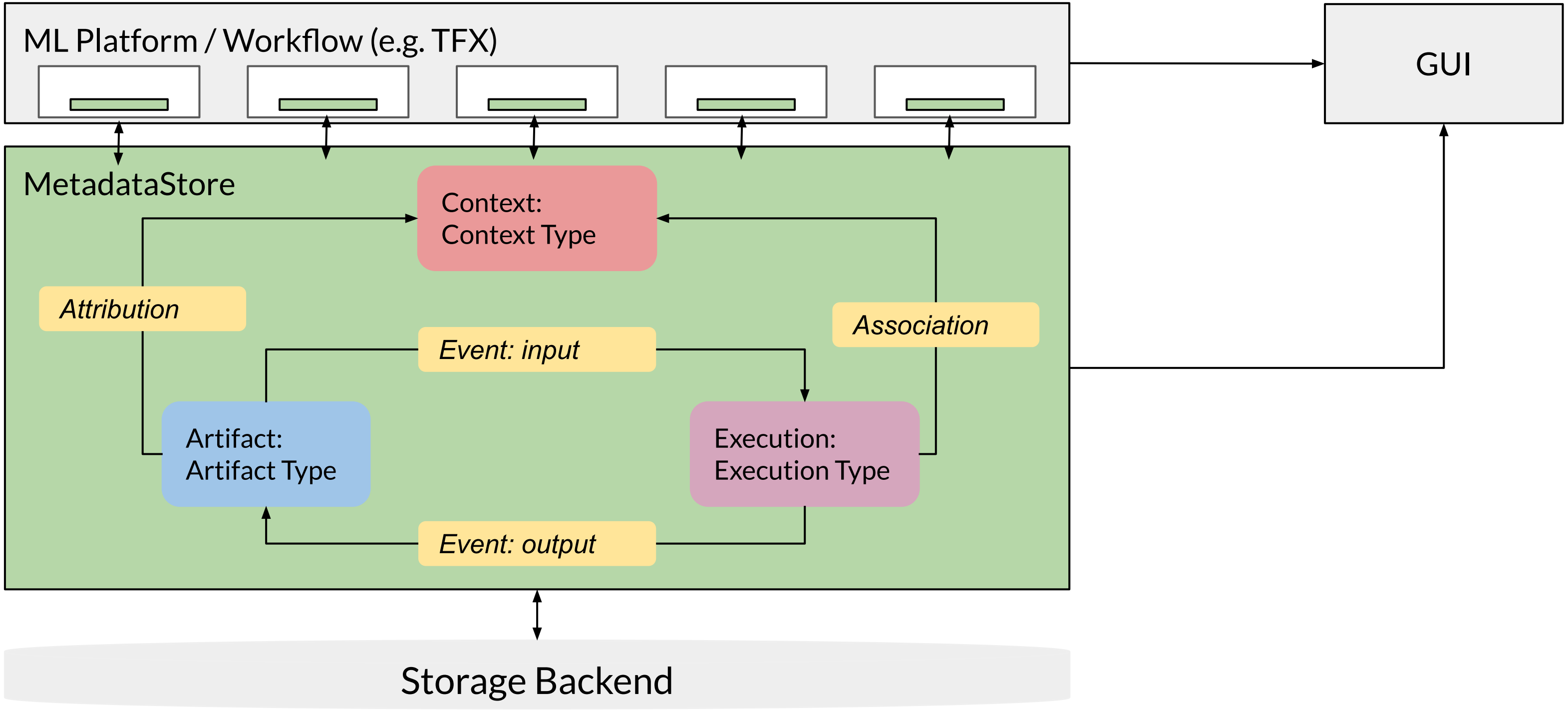
Artifacts: Data going as input or generated as output by a component

Execution: Record of component in pipeline.

Context: Conceptual grouping of executions and artifacts.



Inside MetadataStore



Key points

ML metadata:

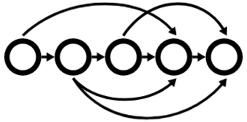
- Architecture and nomenclature
- Tracking metadata flowing between components in pipeline

DeepLearning.AI

Data Journey and Data Storage

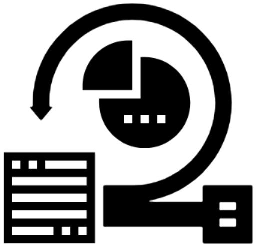
ML Metadata in action

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)
```

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

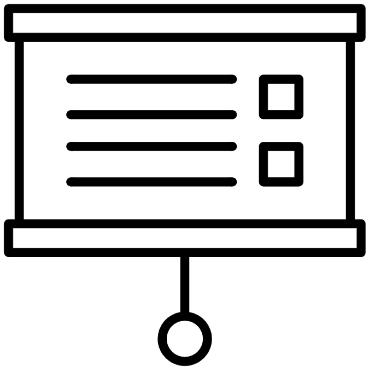
DeepLearning.AI

Evolving Data

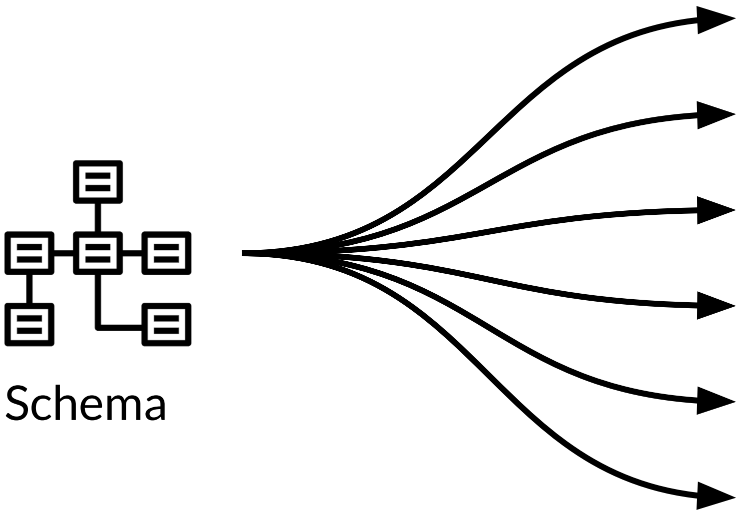
Schema Development

Outline

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



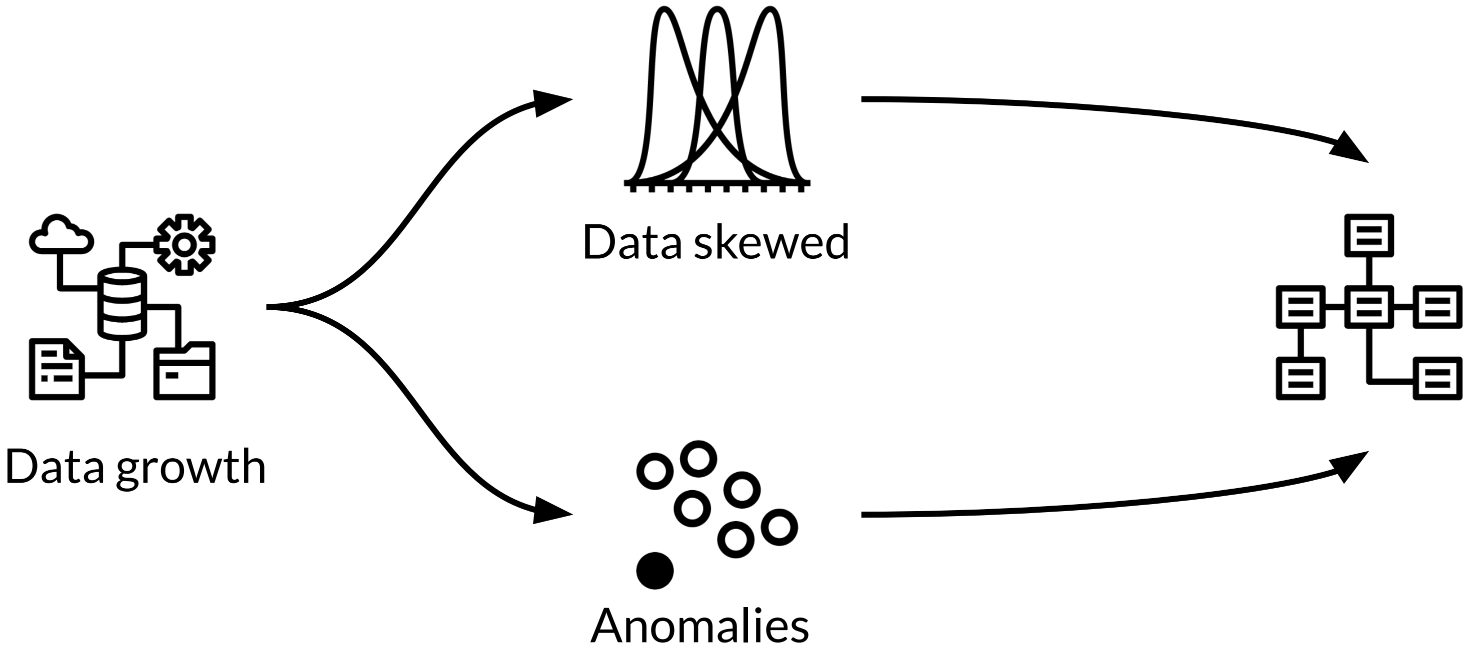
Review: Recall Schema



Schema

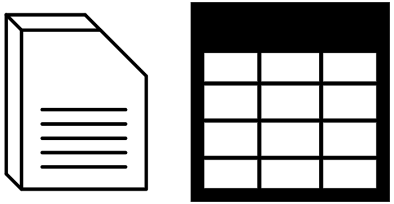
- Feature name
- Type: float, int, string, etc
- Required or optional
- Valency (features with multiple values)
- Domain: range, categories
- Default values

Iterative schema development & evolution

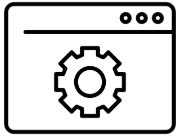


Reliability during data evolution

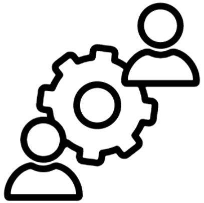
Platform needs to be resilient to disruptions from:



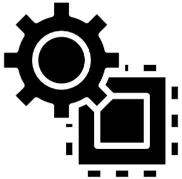
Inconsistent data



Software



User configurations

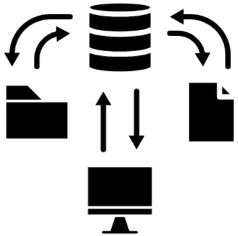


Execution environments

Scalability during data evolution

Platform must scale during:

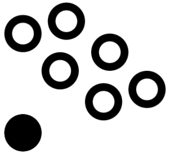
High data volume during training



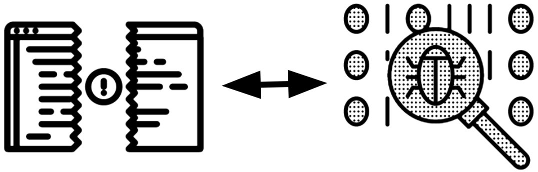
Variable request traffic during serving

Anomaly detection during data evolution

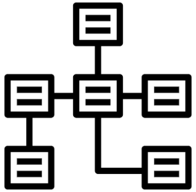
Platform designed with these principles:



Easy to detect anomalies

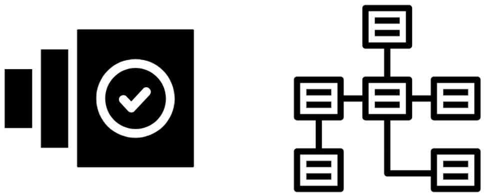


Data errors treated
same as code bugs

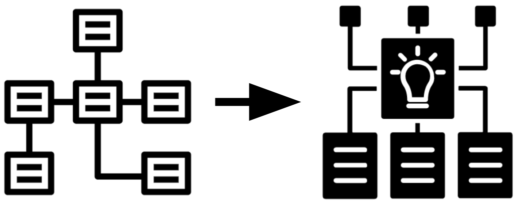


Update data schema

Schema inspection during data evolution



Looking at schema versions to
track data evolution



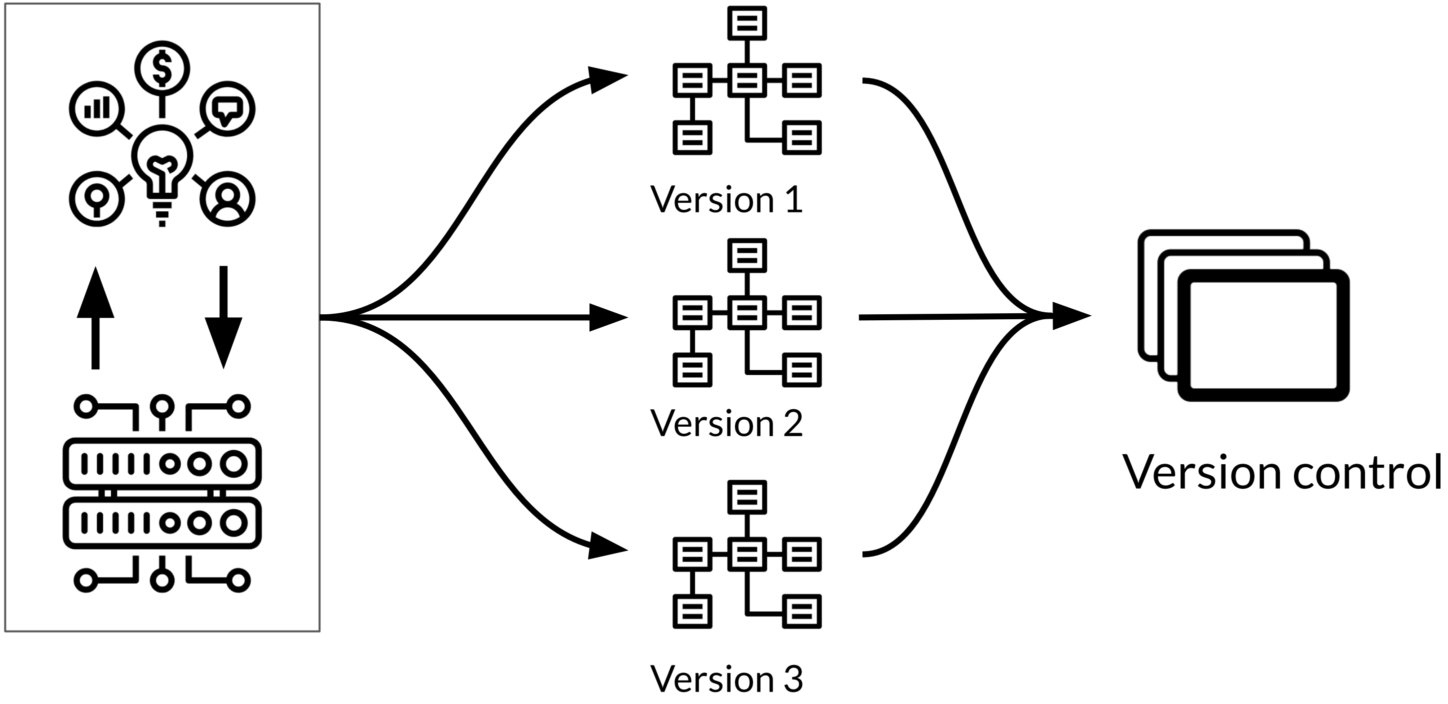
Schema can drive other
automated processes

DeepLearning.AI

Evolving Data

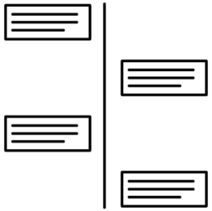
Schema Environments

Multiple schema versions

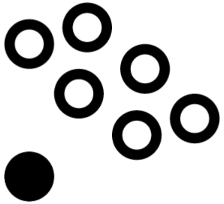


Maintaining varieties of schema

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

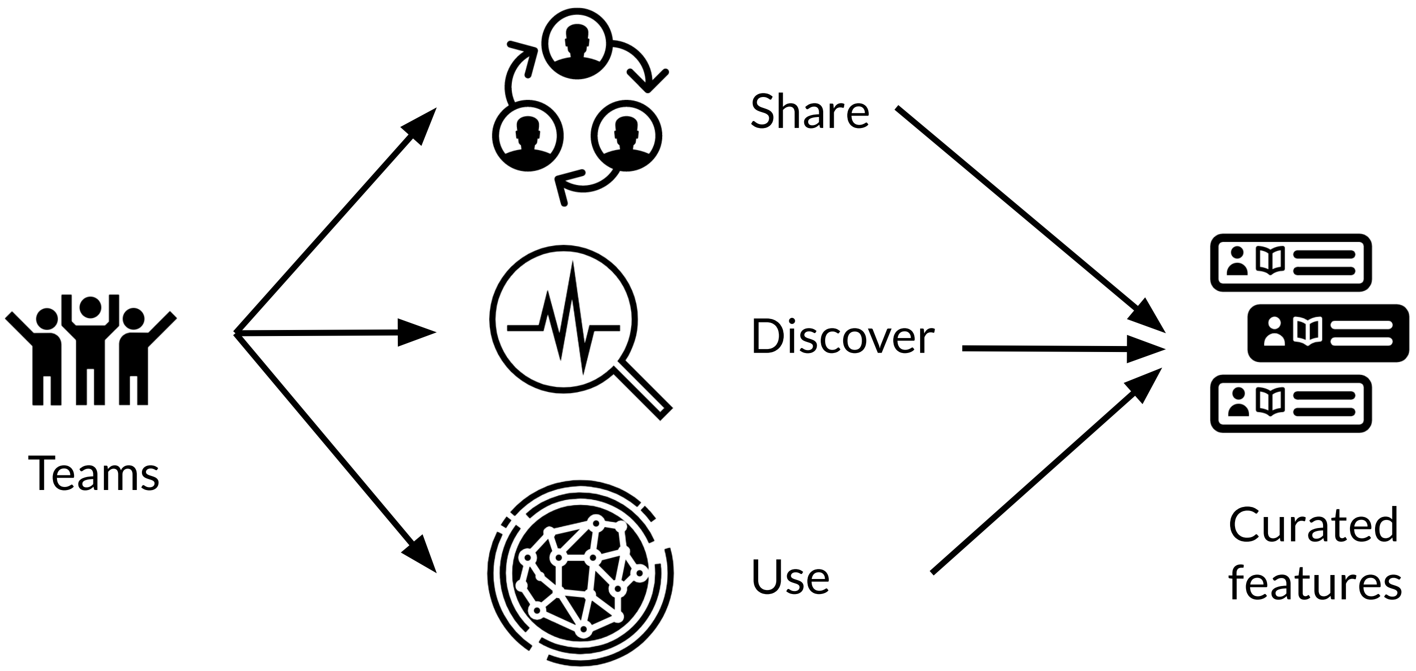
- Iteratively update and fine-tune schema to adapt to evolving data
- How to deal with scalability and anomalies
- Set schema environments to detect anomalies in serving requests

DeepLearning.AI

Enterprise Data Storage

Feature Stores

Feature stores



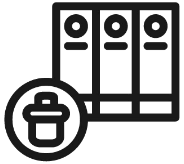
Feature stores

Many modeling problems use identical or similar features

Feature stores

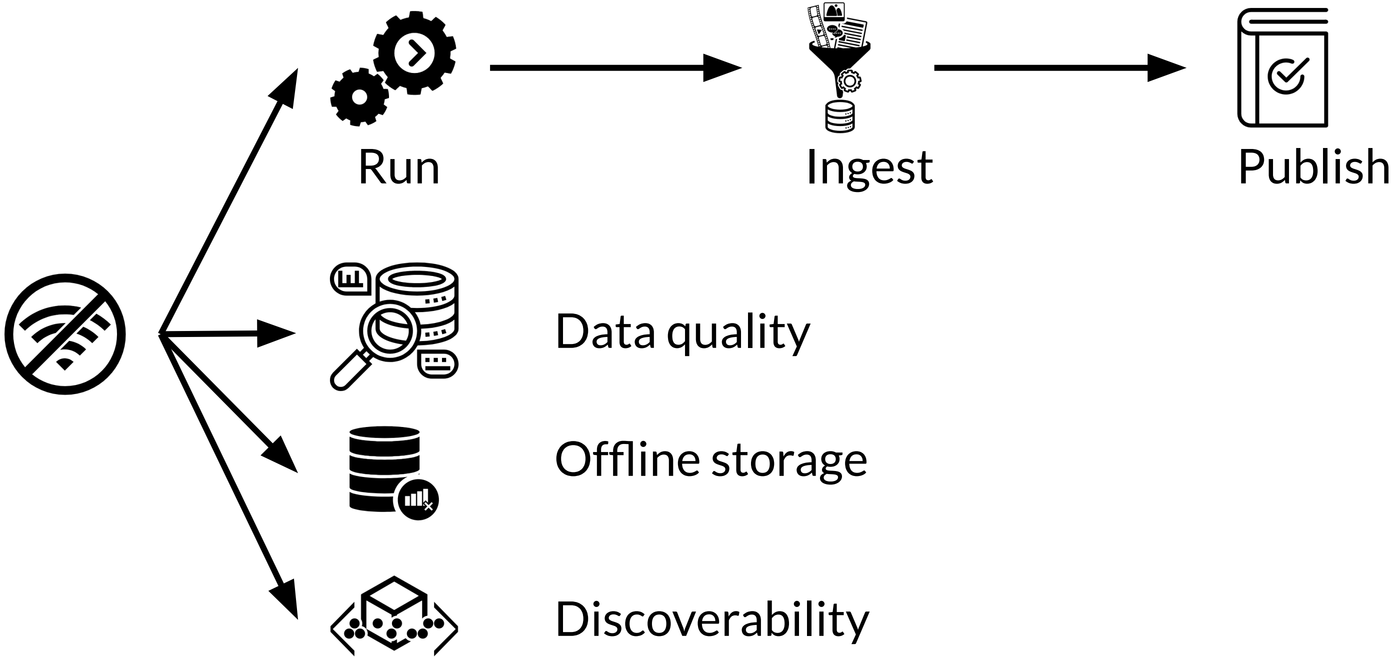
Avoid duplication

Control access

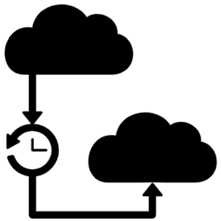


Purge

Offline feature processing



Online feature usage



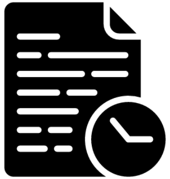
Low latency access to features

Features difficult to compute online

Precompute and store for low latency access

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

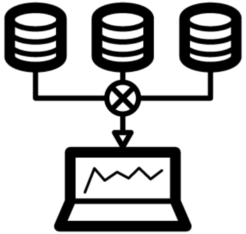
- Managing feature data from a single person to large enterprises.
- Scalable and performant access to feature data in training and serving.
- Provide consistent and point-in-time correct access to feature data.
- Enable discovery, documentation, and insights into your features.

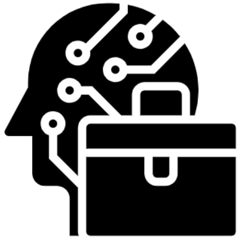
DeepLearning.AI

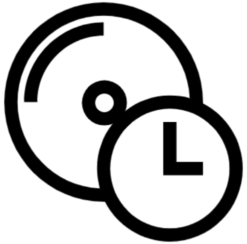
Enterprise Data Storage

Data Warehouse

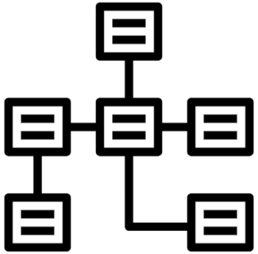
Data warehouse

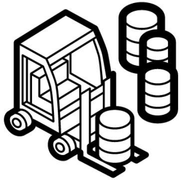

Aggregates
data sources


Processed
and analyzed

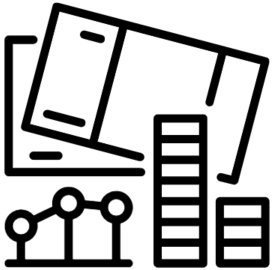

Read
optimized


Not
real time

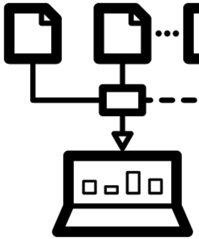

Follows
schema



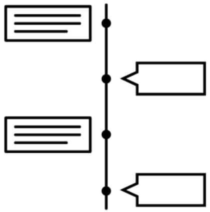
Key features of data warehouse



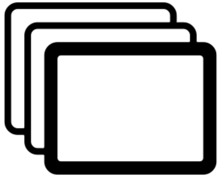
Subject oriented



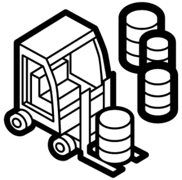
Integrated



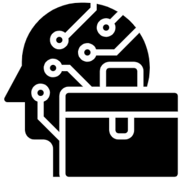
Non volatile



Time variant

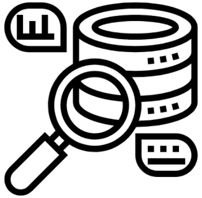


Advantages of data warehouse



Enhanced ability to analyze data

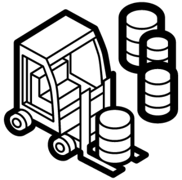
Timely access to data



Enhanced data quality and consistency

High return on investment

Increased query and system performance



Comparison with databases

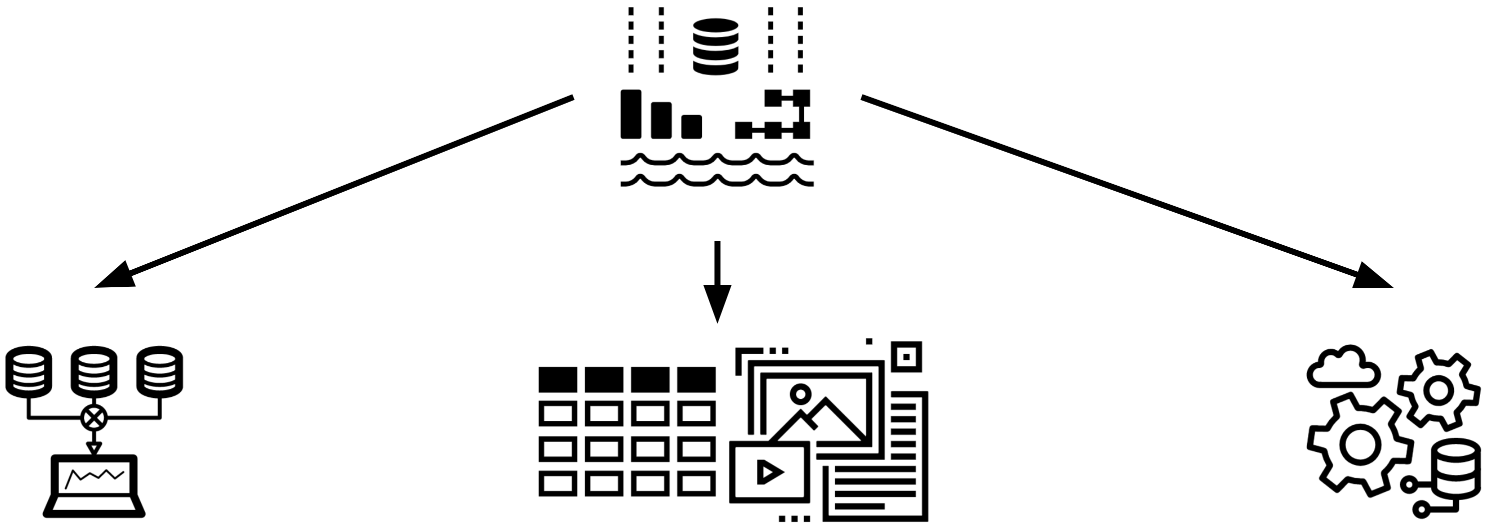
Data warehouse	Database
Online analytical processing (OLAP)	Online transactional processing (OLTP)
Data is refreshed from source systems	Data is available real-time
Stores historical and current data	Stores only current data
Data size can scale to >= terabytes	Data size can scale to gigabytes
Queries 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

DeepLearning.AI

Enterprise Data Storage

Data Lakes

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.