



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Phases of Data

Pravin Y Pawar

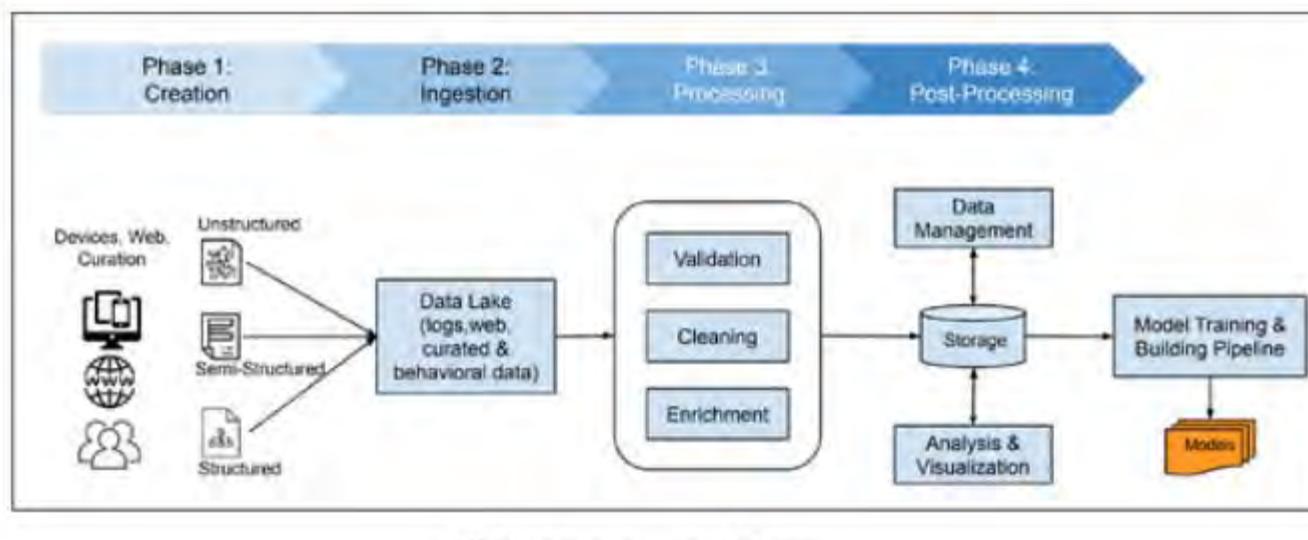
Adapted from
Reliable Machine Learning
By Chen, Murphy, Sculley, Underwood

Data Management

- The data management stage is about transforming the data into a format and storage organization suitable for using it for later stages of the process
- During this process
 - may apply a range of data transformations that are model specific prepare the data for training
 - next operations on the data will be to train ML models, anonymize certain sensitive items of data
 - delete data when we no longer need it or are asked to do so
- To prepare for these operations on the data,
 - take input from the business leaders to answer questions about primary use cases for the data
 - explore possible areas for future exploration

Phases of Data

- Data management in modern ML environments consists of **multiple phases** before feeding the data into model-training pipelines
 - Creation
 - Ingestion
 - Processing (which includes validation, cleaning, and enrichment)
 - Post-processing (which includes data management, storage, and analysis)



Phases of Data(2) - Creation

Sources

- Odd or obvious to state, but ML training data comes from **somewhere**
 - Datasets are all created at some point via some process
 - The process of generating or capturing the data in some data storage system but not our data storage system
 - Examples - logs from serving systems, large collections of images captured from an event, diagnostic data from medical procedures
- Some datasets work well when they are **static** (or at least don't change quickly), and others are useful only when **frequently updated**
- For example,
 - A photo-recognition dataset, may be usable for many months, as long as it is suitably representative of the types of photos would like to recognize with model.
 - On the other hand, if the outside photos represent only winter environments from a temperate climate, the distribution of the photo set will be significantly diversified

Phases of Data(3) - Creation

Kinds of data

- Structured data
 - quantitative with a predefined **data model/schema**, highly organized, and stored in tabular formats like spreadsheets or relational databases
 - can be easily processed by relatively simple code using obvious heuristics.
- Unstructured data
 - qualitative **without a standard data model/schema**
 - cannot be processed and analyzed using conventional data methods and tools
- Semi-structured data
 - doesn't have a specific data model/schema but includes tags and semantic markers
 - a type of structured data that lies between structured and unstructured data
- The character of the internal structure of the data will have significant implications for the way we process, store, and use it.



STRUCTURED



Clear schema, highly organized

E.g. Name, address, geolocation, dates, contact & payment information etc

SEMI-STRUCTURED



Loose schema, tags and semantic markers

E.g. Email content by from, to, inbox, sent, drafts; social media posts with hashtags etc

UNSTRUCTURED



No schema, qualitative

E.g. Email body text, product descriptions, web text, speech, video, images etc

Phases of Data(4) - Creation

Dataset augmentation

- Small amount of training data is available - not enough to train a high-quality model on
- then need to augment that data
 - Tools are available that can do so
- For example, [Snorkel](#)
 - is one good place to start, as it allows us to easily expand a small dataset into a large one
 - has good evidence that this approach can yield extremely good results at low cost, although it does need to be used with caution

Phases of Data(5) - Ingestion

Filter / Selection / Sampling

- The data needs to be received into the system and written to storage for further processing
 - not all data created may be ingested or collected, because we don't want or need to
 - a filtering and selection step necessarily occurs
- Sampling
 - may also simply sample at this stage if have so much data which can not be processed all of it
 - as ML training and other data processing is often extremely computationally expensive
 - an effective way to save money on intermediate processing and training costs
 - important to measure the quality cost of sampling and compare that to the savings
 - is likely to occasionally lose some detail for some events – **unavoidable!**
- In general, ML training systems perform better with more data. Any reduction in data may well impact quality at the same time as reducing costs.

Phases of Data(6) - Ingestion

Mechanism

- Depending on the volume of data and the complexity of our service, the ingestion phase
 - may be as simple as “dump some **files** in that directory over there”
 - or as sophisticated as a **remote procedure call (RPC) endpoint** that receives specifically formatted files and confirms a reference to the data bundle that was received so that its progress through the system can be tracked
- Reliability concerns during the ingestion phase typically focus on **correctness and throughput**
 - Correctness is the general property that the data is properly read and written in the correct place without being unnecessarily skipped or misplaced
 - **Monitoring the existence and condition of data before and during ingestion is the most difficult part of the data pipeline.**

Phases of Data(7) - Processing

Validation

- Set of common operations to make the data ready for training
 - Validation
 - cleaning and ensuring data consistency
 - enriching and extending
- Validation
 - Efficient and powerful ML models can never do what we want them to do with bad data
 - In production, a common reason for errors in the data is bugs in the code that is collecting the data in the first place
 - Data ingested from external sources might have a lot of errors
 - **extremely important to validate the incoming data especially when there is a well-defined schema and/or an ability to compare against the last known valid feed**
 - performed against a common definition of the field
 - need to both store and be able to reference those standard definitions

Phases of Data(8) - Processing

Cleaning and ensuring data consistency

- Even with a decent validation framework in place, most **data is still messy**
 - may have **missing fields, duplicates, misclassifications, or even encoding errors**
 - more data we have, the more likely that cleaning and data consistency will be its own stage of processing
- **ML pipeline** will assuredly have **code whose job is to clean the data**
 - can either put this code in a single place, where it can be reviewed and improved,
 - or put aspects of it throughout the training pipeline
- Another set of data consistency tasks during this portion is the **normalization of data**
 - refers to a set of techniques used to transform the input data into a **similar scale**
- Common technique is that of putting the data into **buckets**
 - map a range of data into a much smaller set of groups that represent the same range

Phases of Data(9) - Processing

Enriching and extending

- Enriching
 - Combine data with data from other sources
- Labeling
 - most common and fundamental way to extend the data
 - identifies a given event or record by bringing in confirmation from an outside source of data (sometimes a human)
 - Labeled data is the key driver of all supervised ML and is often one of the most challenging and expensive parts of the whole ML process
- Joins
 - Might use many external data sources to extend training data

Phases of Data(10) - Post-processing

Storage

- Need to store the data somewhere
 - How and where we store the data is mostly driven by how we tend to use it
 - which is really a set of questions about training and serving systems
- Two predominant concerns here:
 - efficiency of storage
 - metadata
- The efficiency of a storage system is driven by access patterns
 - that are driven by the model structure, team structure, and training process
- Metadata helps humans interact with the storage
 - When multiple people work on building models on the same data (or when the same person works on this over time), metadata about the stored features provides huge value
 - Roadmap to understanding how the last model was put together and how we might want to put together the model
 - More commonly undervalued parts of an ML system.

Phases of Data(11) - Post-processing

Management

- Data management system is primarily motivated by two factors
 - The business purpose to which we intend to put the data
 - What problem are we trying to solve?
 - What is the value of that problem to our organization or our customers?
 - The model structure and strategy
 - What models do we plan to build?
 - How are they put together?
 - How often are they refreshed?
 - How many of them are there?
 - How similar to one another are they?
- Every choice we make about our data management system is constrained by, and constrains, these two factors!
- If data management is about how and why we write data, ML training pipelines are about how and why we read it.

Phases of Data(12) - Post-processing

Analysis and Visualization

- Process of transforming large amounts of data into an easy-to-navigate representation using statistical and/or graphical techniques and tools
 - Essential task of ML architectures and knowledge discovery techniques to make data less confusing and more accessible
- Not enough just to show a pie chart or bar graph
 - Need to provide the human reader an explanation of
 - what each record in the dataset means,
 - how it is linked with the records in other datasets
 - and whether it is clean and safe to use for training models
- Practically impossible for data scientists to look at large datasets without well-defined and high-performant data visualization tools and processes



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Data Management Principles

Pravin Y Pawar

Adapted from
Reliable Machine Learning
By Chen, Murphy, Sculley, Underwood

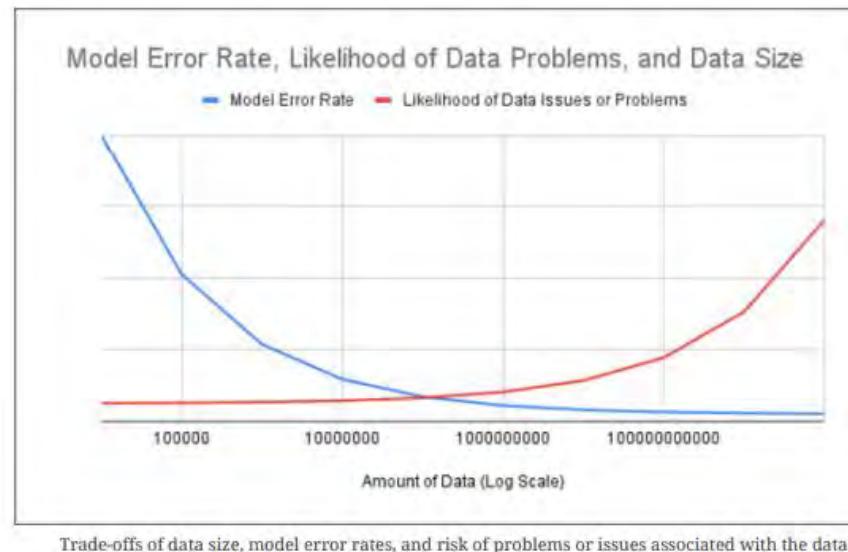
Data and ML Pipelines

Relation

- ML systems are data processing pipelines
 - with a purpose to extract usable and repeatable insights from data
 - fundamentally, they consume data, and output a processed representation of that data
- Key differences between ML pipelines and conventional log processing or analysis pipelines
 - have some very different and specific constraints and fail in different ways
 - success is hard to measure,
 - many failures are difficult to detect
 - depend thoroughly and completely on the structure, performance, accuracy, and reliability of their underlying data systems
- Overwhelmingly interested in two things:
 - the data used to construct the models
 - the processing pipeline that takes the data and transforms it into models

Data as Asset

- Unarguably data is an **important asset** in ML systems
 - certainly impossible to have an ML system without data
 - often true that a simple (or even simplistic) ML system with more (and higher-quality) training data can outperform a more sophisticated system with less, or less representative, data



- Organizations continue to scramble to collect as much data as possible, hoping to find ways to turn that data into value
 - many organizations have made this into a profoundly successful business model
 - Netflix, whose ability to recommend high-quality shows and movies to customers was an early differentiator

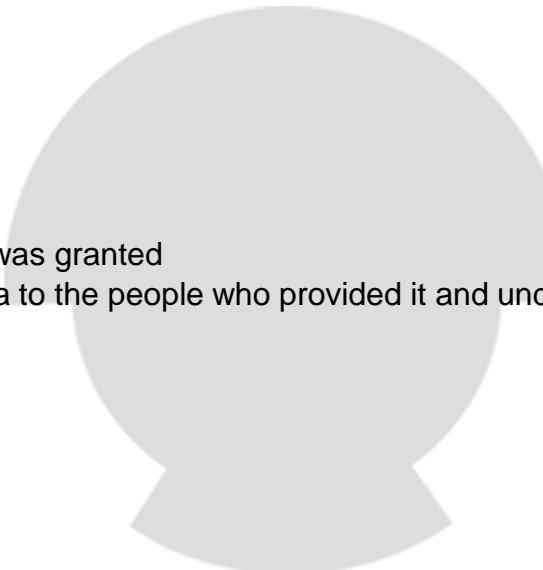
Data as Liability

- Anything could be an asset, under the right (wrong) circumstances, it can also be a liability
 - For data, the acquisition, collection, and curation of data can expose areas of unexpected nuance and complexity in the data
 - can lead to potential harm for us and for our users
- The intent is to enumerate enough of the complexity to dissuade any readers from
 - simply thinking “more data == better” or
 - thinking “this stuff is easy”

Data as Liability(2)

Questions with data collection

- Data must be collected in compliance with applicable laws, which might be based on
 - where organization is located
 - where the data originates
 - on organizational policies
- Significant restrictions on
 - what counts as data about people
 - how to get permission to store the data
 - how to store and retrieve the permission that was granted
 - whether we need to provide access to the data to the people who provided it and under what circumstances
- Restrictions may come from
 - laws
 - industry practices
 - insurance regulations
 - corporate governance policies
 - or any range of other sources
- For example, some jurisdictions include prohibition against
 - collecting personally identifiable information (PII) about an individual without their explicit written consent
 - along with the requirement to delete that data upon request by the data subject



Not technical question, but policy question!

Data as Liability(3)

Questions with data storage

- Data must be secured from external and internal access
 - Employees should not be able to view or change private user data without restriction and without detailed logging of that
- Pseudonymization
 - approach to reducing data access and reducing the auditing surface is to anonymize the data
 - private identifiers are replaced with others in a reversible fashion, and reversing the pseudonymization requires access to an additional data or system
 - protects the data from casual inspection by engineers working on the pipeline but permits discovery if we find that we need to reverse the anonymization
 - preserves the properties of data that are relevant to ML model
- A better approach is permanently removing any direct connection between private data about a person and the data that we use to train on
 - substantially reduce the risk of the data and increase the flexibility to handle it
 - **harder than it seems!**
 - Doing this in a way that's not trivially reversible but still valuable can be difficult

Data as Liability(4)

Questions with data deletion

- Might need to delete data
 - at the request of individual users, local laws, regulations, compliances
 - in cases where we no longer have permission to store the data
- Deleting data and having it actually be deleted is surprisingly hard!
 - been true since at least the early MS-DOS days
 - today's computing environment makes deletion even harder due to multiple copies of the data and metadata management
- Important to be certain that people want their data deleted without putting up arbitrary barriers
 - impose a short delay before really deleting the data
 - also underlying data structures, indices, and backups may be reconstructed

The Data Sensitivity of ML Pipelines

- ML pipelines are unusually sensitive to their input data compared to most other data processing pipelines
 - All data processing pipelines are subject to the correctness and volume of their input data
 - ML pipelines are furthermore sensitive to subtle changes in distribution of the data
 - can easily go from mostly right to significantly wrong simply by omitting a small fraction of the data
- For example, consider a real-world system that somehow loses all of the data from a particular country, region, or language
 - drop all of the data from December 31 of a given year, lose the ability to detect New Year's Eve shopping trends
 - treads may be substantially different from the surrounding days in December and January
 - results into systematically biased results in significant confusion in models' understanding and predictions
- As a result of this sensitivity, the ability to aggregate, process, and monitor data is critical to successfully managing ML data pipelines.

Data Reliability

- Because our data processing system needs to work, the data must have several characteristics as it traverses the system
- Needs to know basics of making sure that
 - the data is not lost (durability)
 - is the same for all copies (consistency)
 - is tracked carefully as it changes over time (version control)
 - is read fast enough (performance)
 - is ready to be read (availability)



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Data Management and its Components

Pravin Y Pawar

Data Management

What and Why?

- Data management is the **practice of ingesting, processing, securing and storing an organization's data**,
 - where it is then utilized for strategic decision-making to improve business outcomes
- Over the last decade, developments **within hybrid cloud, artificial intelligence, the Internet of Things (IoT), and edge computing** have led to the exponential **growth of big data**
 - creating even more complexity for enterprises to manage
- A data management discipline within an organization has become **an increasing priority** as this growth has created significant challenges,
 - such as **data silos, security risks, and general bottlenecks to decision-making**
- Teams address these challenges head on with a number of **data management solutions**, which are **aimed to clean, unify, and secure data**
 - Allows leaders to glean insights through dashboards and other data visualization tools, enabling informed business decisions
 - Empowers data science teams to investigate more complex questions, allowing them to leverage more advanced analytical capabilities

Data Management Framework

Components



[Source : trellance](#)

Data Management Framework

Components : described

- Data Governance
 - provides the overarching support to data management through stewardship, policies, processes, standards, and adherence to leading practices
- Data Architecture
 - provides the infrastructure for the storage, integration, and use of data throughout the organization
- Metadata
 - allows to use data more efficiently by providing critical information about data attributes
- Data Quality
 - provides the structure necessary to have data that fulfills the needs of the business
- Data Lifecycle
 - follows the data throughout the company, providing integrity from the initial introduction into the company through the final deletion from the company
- Analytics
 - applies statistical and visualization techniques that lead to valuable insights that can help the company make better business decisions.
- Data Privacy
 - supports the needs of the business to share data internally and externally.

Data Management Components

- The scope of a data management discipline is quite broad!
- A strong data management strategy typically implements the following components to streamline their strategy and operations throughout an organization:
 - Data processing
 - Data storage
 - Data governance
 - Data security

Data Management Components(2)

Data processing

- In this stage,
 - Raw data is ingested from a range of data sources, such as web APIs, mobile apps, Internet of Things (IoT) devices, forms, surveys, and more
 - Then, usually processed or loaded, via data integration techniques, such as extract, transform, load (ETL) or extract, load, transform (ELT)
 - ETL has historically been the standard method to integrate and organize data across different datasets
 - ELT has been growing in popularity with the emergence of cloud data platforms and the increasing demand for real-time data
- Independently of the data integration technique used,
 - the data is usually filtered, merged, or aggregated during the data processing stage to meet the requirements for its intended purpose,
 - which can range from a business intelligence dashboard to a predictive machine learning algorithm

Data Management Components(3)

Data storage

- Data can be stored before or after data processing, the type of data and purpose of it will usually dictate the storage repository that is leveraged
- Data warehousing
 - requires a defined schema to meet specific data analytics requirements for data outputs,
 - such as dashboards, data visualizations, and other business intelligence tasks
 - which are usually directed and documented by business users in partnership with data engineers,
 - who will ultimately execute against the defined data model
 - underlying structure of a data warehouse is typically organized as a relational system (i.e. in a structured data format),
 - sourcing data from transactional databases
- Other storage systems, such as data lakes,
 - incorporate data from both relational and non-relational systems, becoming a sandbox for innovative data projects
 - benefit data scientists in particular, as they allow them to incorporate both structured and unstructured data into their data science projects

Data Management Components(4)

Data governance

- A set of standards and business processes which ensure that data assets are leveraged effectively within an organization
 - includes processes around data quality, data access, usability, and data security
- Data governance councils
 - tend align on taxonomies to ensure that metadata is added consistently across various data sources.
 - help to define roles and responsibilities to ensure that data access is provided appropriately; this is particularly important to maintain data privacy.

Data Management Components(5)

Data security

- Data security sets guardrails in place to protect digital information from unauthorized access, corruption, or theft
 - more scrutiny is placed upon the security practices of modern businesses
 - to ensure that customer data is protected from cybercriminals or disaster recovery incidents
- While data loss can be devastating to any business, data breaches, in particular, can reap costly consequences from both a financial and brand standpoint
- Data security teams can better secure their data by leveraging encryption and data masking within their data security strategy.

Data Platform

- Data management tools **require a robust data and computing infrastructure**
 - that scales to meet business demand without impacting performance
- Most companies now build a **data lake to store raw data** of any type from any system, often in **Hadoop**
 - but increasingly in a **public cloud** (e.g., Microsoft Azure)
- Companies also build
 - data warehouses to support performance reporting (e.g., reports, dashboards, and light analysis) on structured data
 - temporary sandboxes for testing, analysis, and prototyping purposes
- Data environments require
 - fast query processing
 - large storage capacity
 - elastic computing
 - In-memory caches
 - massively parallel query processing
 - columnar storage
 - and large in-memory data grids and computing clusters

Reference:

What is data management?

<https://www.ibm.com/in-en/topics/data-management>



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Data Formats

Pravin Y Pawar

Adapted from Designing Machine Learning Systems
by Chip Huyen

Data Sources

- An ML system can work with data from many different sources
 - have different characteristics with different access patterns
 - can be used for different purposes
 - and require different processing methods
- Understanding the sources of data can help access and manipulate data more efficiently

Data Sources(2)

User Generated data

- One common source is user input data, data explicitly input by users, which is often the input on which ML models can make predictions
 - can be texts, images, videos, uploaded files, etc.
- If there are wrong way for humans to input data, humans are going to do it, and as a result, user input data can be easily mal-formatted
- If user input is supposed to be
 - texts, they might be too long or too short
 - numerical values, users might accidentally enter texts
 - file uploading, they might upload files in the wrong formats
- Challenges
 - User input data requires more heavy-duty checking and processing
 - Users also have little patience. In most cases, when we input data, we expect to get results back immediately
 - Therefore, user input data tends to require fast processing.

Data Sources(3)

System-generated data

- Data generated by different components of systems, which include various types of logs and system outputs such as model predictions
- Logs
 - Can record the state of the system and significant events in the system, such as memory usage, number of instances, services called, packages used, etc. Can record the results of different jobs, including large batch jobs for data processing and model training
 - Provides visibility into how the system is doing, and the main purpose of this visibility is for debugging and possibly improving the application
 - Much less likely to be mal-formatted as its system generated
 - Don't need to be processed as soon as they arrive
 - acceptable to process logs periodically, such as hourly or even daily
 - might still want to process logs fast to be able to detect and be notified whenever something interesting happens
- Because debugging ML systems is hard,
 - Common practice to log everything you can
 - Means volume of logs can grow very, very quickly
- Leads to two problems
 - The first is that it can be hard to know where to look because signals are lost in the noise
 - Many services that process and analyze logs, such as Logstash, DataDog, Logz, etc.
 - The second problem is how to store a rapidly growing amount of logs
 - In most cases, store logs for as long as they are useful, and can discard them when they are no longer relevant
 - Can also be stored in low-access storage that costs much less than higher-frequency-access storage

Data Sources(4)

System-generated users data

- Users' behaviors
 - such as clicking, choosing a suggestion, scrolling, zooming, ignoring a popup, or spending an unusual amount of time on certain pages
 - system-generated data, it's still considered part of user data
 - might be subject to privacy regulations
 - Can be used for ML systems to make predictions and to train their future versions
- Internal databases
 - generated by various services and enterprise applications in a company
 - manage their assets such as inventory, customer relationship, users, and more
 - can be used by ML models directly or by various components of an ML system
 - For example,
when users enter a search query on Amazon, one or more ML models will process that query to detect the intention of that query — what products users are actually looking for? — then Amazon will need to check their internal databases for the availability of these products before ranking them and showing them to users.

Data Sources(5)

Third-party data - wonderfully weird world

- Types of data
 - First-party data is the data that company already collects about users or customers
 - Second-party data is the data collected by another company on their own customers that they make available to you
 - Third-party data companies collect data on the public who aren't their customers
- The rise of the Internet and smartphones has made it much easier for all types of data to be collected
 - Riddled with privacy concerns
 - Data from apps, websites, check-in services, etc. are collected and (hopefully) anonymized to generate activity history for each person
 - Third-party data is usually sold as structured data after being cleaned and processed by vendors.

Data Formats

Need

- Once have data, need to store data (or “persist” it, in technical terms)!
- Since data comes from multiple sources with different access patterns
 - storing your data isn’t always straightforward and can be costly
 - important to think about how the data will be used in the future so that the format will make sense
- Questions needs to be considered
 - How do I store multimodal data?
 - When each sample might contain both images and texts?
 - Where to store your data so that it’s cheap and still fast to access?
 - How to store complex models so that they can be loaded and run correctly on different hardware?
- Data serialization
 - The process of converting a data structure or object state into a format that can be stored or transmitted and reconstructed later
 - Many data serialization formats
- When considering a format to work with, need to consider different characteristics such as
 - human readability
 - access patterns
 - and whether it’s based on text or binary, which influences the size of its files

Data Formats(2)

Common data formats

Format	Binary/Text	Human-readable	Example use cases
JSON	Text	Yes	Everywhere
CSV	Text	Yes	Everywhere
Parquet	Binary	No	Hadoop, Amazon Redshift
Avro	Binary primary	No	Hadoop
<u>Protobuf</u>	Binary primary	No	Google, TensorFlow (<u>TFRecord</u>)
Pickle	Binary	No	Python, <u>PyTorch</u> serialization

Common data formats and where they are used.

Data Formats(3)

JSON

- JSON, JavaScript Object Notation, is everywhere
 - Though it was derived from JavaScript, it's language-independent — most modern programming languages can generate and parse JSON
 - Human-readable
 - Key-value pair paradigm is simple but powerful, capable of handling data of different levels of structuredness
- For example, persons data can be stored in a structured format like the following

```
{  
  "firstName": "Boatie",  
  "lastName": "McBoatFace",  
  "isVibing": true,  
  "age": 12,  
  "address": {  
    "streetAddress": "12 Ocean Drive",  
    "city": "Port Royal",  
    "postalCode": "10021-3100"  
  }  
}
```

- The same data can also be stored in an unstructured blob of text like the following

```
{  
  "text": "Boatie McBoatFace, aged 12, is vibing, at 12 Ocean Drive, Port Royal,  
  10021-3100"  
}
```

Data Formats(4)

Row-major vs. Column-major Format

- Two common formats that represents distinct paradigms are CSV and Parquet
 - CSV is row-major, which means consecutive elements in a row are stored next to each other in memory
 - Parquet is column-major, which means consecutive elements in a column are stored next to each other
- Modern computers process sequential data more efficiently than non-sequential data
 - If a table is row-major, accessing its rows will be faster than accessing its columns in expectation
 - For row-major formats, accessing data by rows is expected to be faster than accessing data by columns
- Column-major formats allow flexible column-based reads, especially if data is large with thousands, if not millions, of features
 - Row-major formats allow faster data writes
- Overall,
 - row-major formats are better when you have to do a lot of writes
 - column-major ones are better when you have to do a lot of column-based reads.

Data Formats(5)

Row-major vs. Column-major Format

Row-major:

- data is stored and retrieved row-by-row
- good for accessing samples

Column-major:

- data is stored and retrieved column-by-column
- good for accessing features

	Column 1	Column 2	Column 3
Example 1
Example 2
Example 3

Data Formats(6)

Text vs. Binary Format

- CSV and JSON are text files whereas Parquet files are binary files
 - Text files are files that are in plain text, which usually mean they are human-readable
 - Binary files are files that contain 0's and 1's, and meant to be read or used by programs that know how to interpret the raw bytes
 - A program has to know exactly how the data inside the binary file is laid out to make use of the file

- Binary files are more compact.
 - AWS recommends using the Parquet format because “the Parquet format is up to 2x faster to unload and consumes up to 6x less storage in Amazon S3, compared to text formats.”

```
In [2]: df = pd.read_csv("data/interviews.csv")
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17654 entries, 0 to 17653
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Company     17654 non-null   object 
 1   Title       17654 non-null   object 
 2   Job         17654 non-null   object 
 3   Level       17654 non-null   object 
 4   Date         17652 non-null   object 
 5   Upvotes     17654 non-null   int64  
 6   Offer        17654 non-null   object 
 7   Experience   16365 non-null   float64
 8   Difficulty   16376 non-null   object 
 9   Review       17654 non-null   object 
dtypes: float64(1), int64(1), object(8)
memory usage: 1.3+ MB
```

```
In [3]: Path("data/interviews.csv").stat().st_size
Out[3]: 14200063
```

```
In [4]: df.to_parquet("data/interviews.parquet")
Path("data/interviews.parquet").stat().st_size
Out[4]: 6211862
```



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Data Models

Pravin Y Pawar

Adapted from Designing Machine Learning Systems
by Chip Huyen

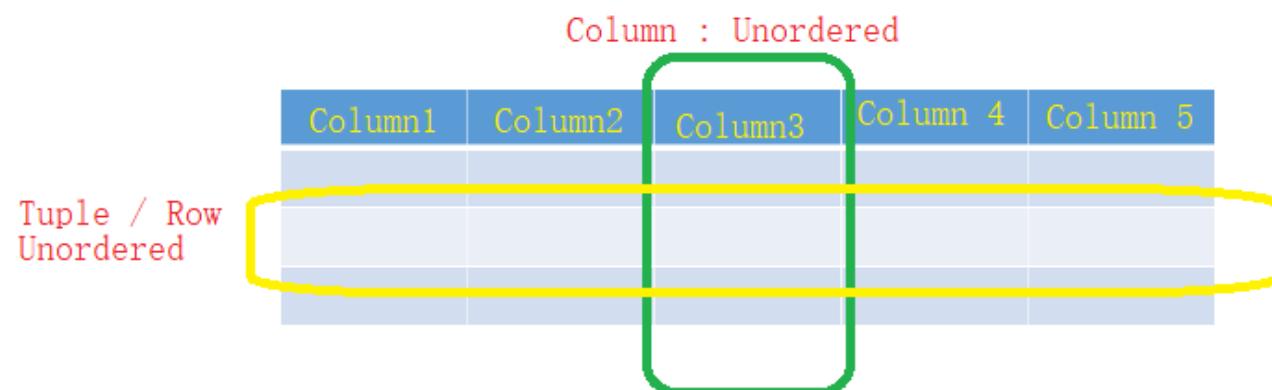
Data Models

- Most important part of developing software, because they have such a profound effect
 - on how the software is written
 - on how we think about solving the problem
- Describes how data is represented in terms of attributes
 - Car can be represented by its make, model, year, color etc., as well by its owner, license plate etc.
 - Most applications are built by layering one data model on top of another
- Many different kinds of data models
 - Every data model embodies assumptions about how it is going to be used
 - Building software is hard enough, even when working with just one data model, and without worrying about its inner workings
- Selecting a data representation affects
 - the ways systems are built
 - The problems systems can solve

Relational Model

Most persistent ideas in computer science

- Invented by Edgar Codd in 1970, but still **going strong today, even getting more popular!**
- **Simple and Powerful**
 - Data is organized into relations; Each relation is a set of tuples
 - Table is visual representation of relation, each row o table makes up a tuple
 - Relations are unordered – can shuffle order of rows or columns
 - Usually stored in CSV or Parquet format



Relational Model(2)

Normalization

- Desirable for relations to be normalized
 - Can follow forms such as first normal form (1NF), second normal form (2NF) etc.
 - Can reduce data redundancy and improve data integrity
- For example, consider Books relation
 - Lot of duplicate data – publisher , country
 - If something change, needs to be reflected everywhere
 - If normalized, updates needs to be managed in single place

Title	Author	Format	Publisher	Country	Price
Book1	Pawar	Paperback	Press1	UK	20
Book1	Pawar	E-book	Press1	UK	10
Book2	Pravin	Paperback	NotedDraft	US	30
Book3	PYP	Paperback	Press1	UK	30
Book2	Pravin	Paperback	NotedDraft	US	15

Title	Author	Format	PubID	Price
Book1	Pawar	Paperback	1	20
Book1	Pawar	E-book	1	10
Book2	Pravin	Paperback	2	30
Book3	PYP	Paperback	1	30
Book2	Pravin	Paperback	2	15

PubID	Publisher	Country
1	Press1	UK
2	NotedDraft	US

Major downside : need to join data back from multiple tables – expensive for large tables!

Relational Model(3)

Querying

- Databases built around relational data model are **relational databases**
- Language used to specify data to be retrieved from database is **query language**
 - Most popular is SQL
- SQL
 - Declarative query language – specify the output needed rather than steps needed to get it
 - Specify the pattern of data –
 - Tables from which data is needed
 - Conditions the results must meet
 - Transformations, such as joins, sort, group, aggregate
 - Database figures out how to provide those results
 - How to break query in parts
 - What methods to user execute each part of query
 - Order in which the parts needs to be executed

NoSQL

Need and Origin

- Relational model generalizes a lot but still restrictive
 - Needs a **strict schema** – schema management is painful!
 - Per Couchbase 2014 survey, frustration with schema management is #1 reason for adoption of non relational databases
- Latest movement against relational data model is **NoSQL**
 - Started as hashtag for meetup to discuss non-relational databases
 - Retroactively reinterpreted as Not Only SQL – many NoSQL data systems supports relational models
- Two major types – **Document and Graph model**
 - Document model targets use cases where data comes in self-contained documents
 - Graph model targeting use cases where relationships between data items are common and important

NoSQL(2)

Document Model

- Built around concept of “**document**”
 - Single continuous string, encoded as JSON, XML or binary format like BSON
 - Each document has a unique key that represents that document, which can be used to retrieve it
 - Collection of document analogous to table in relational database, document analogous to row
 - Can convert relation into a collection of documents
 - Collection of document much more flexible than table
 - All rows in table must follow same schema while documents in same collection can have different schemas
- For example, a book can be represented as document as follows

```
{  
    "Title": "Book1",  
    "Author": "Pravin",  
    "Publisher": "Press1",  
    "Country": "US",  
    "Sold as": [  
        { "format": "Paperback", "Price": "20"},  
        { "format": "E-books", "Price": "10"},  
    ]  
}
```

NoSQL(3)

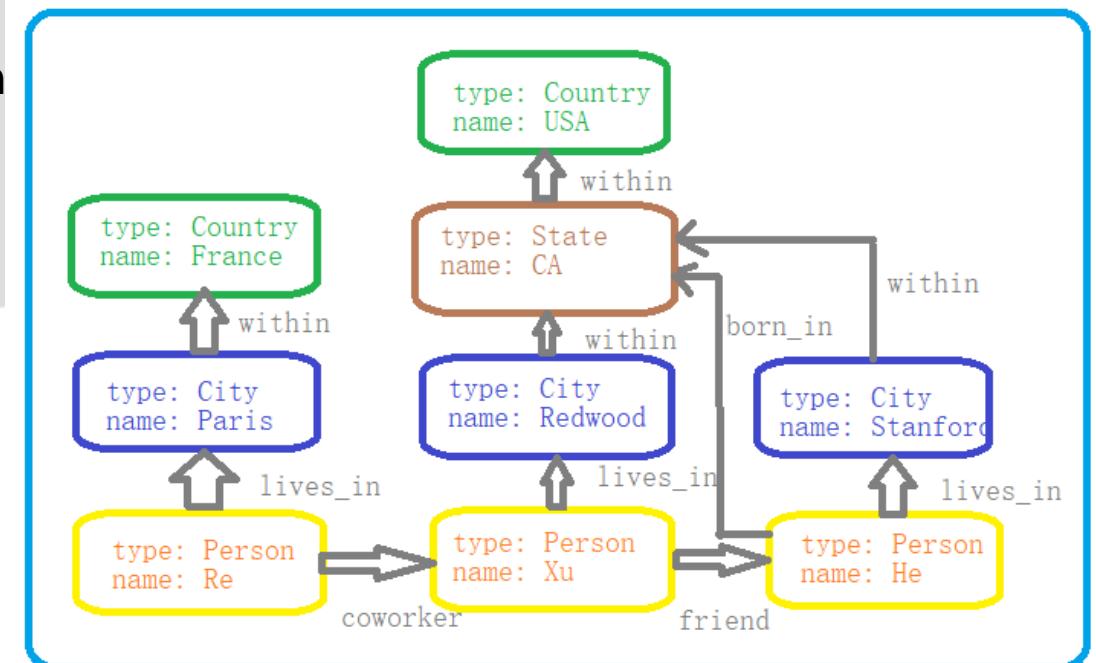
Document Model – Schema less – does not enforce schema

- Misleading!
 - Application that reads documents assumes some kind of structure of documents
 - Document databases just shifts responsibility of assuming structures from application that writes data to the application that reads that data!
- Has better locality than relational model
 - In relational model, data is spread across different tables and needs to be joined when required
 - In document model, all information is stored in single, self-contained place making it easier to read
 - Difficult to execute joins across documents compared to across tables
- Because of different strengths of document and relational data models,
 - Common to use both models for different tasks in same database systems
 - PostgreSQL and MySQL support them both!

NoSQL(4)

Graph Model

- Built around concept of “Graph”
 - Graph consists of nodes and edges, where edges represents relationship between the nodes
 - Database that uses graph structures to store its data is **graph database**
 - In document database, content of each document is priority
 - **In graph database, relationships between data items are priority!**
 - Faster to retrieve data faster as relationships are models in that way
- For example, consider the following representation
 - that may come from social media
 - Find person born in USA
 - **Which one is simpler – graph or relational model?**



Structured vs Unstructured data

Structured data

- Follows a predefined data mode – aka schema
 - For example, Person model has two data values, age of type number and name as string
 - Predefined structure makes data analysis easier
- Disadvantage – commitment to predefined schema!
 - If schema changes, retrospectively update all data – causes mysterious bugs in the process
 - As requirements changes over time,
 - Committing to a predefined schema can be restricting
 - If data coming from multiple sources, don't have any control over schema
 - **Unstructured data becomes appealing!**

Structured vs Unstructured data(2)

Unstructured data

- Does not adhere to predefined schema!
- Usually a text but can also be numbers, dates, images, audio
- Might still contain intrinsic patterns that helps to extract meaningful structures out of data
- Allows for more flexible storage options
 - If storage follows schema, can only store data following that schema
 - If storage does not follows schema, can store any type of data!
 - Can convert all data, regardless of types and formats into bytestrings and store them together
- **Data Warehouse vs Data Lake**
 - Data warehouse is repository to store structured data
 - Data Lake is repository to store unstructured data
 - Data lakes are usually used to store raw data before processing
 - Data warehouses are used to store data that has been processed into formats ready to be used

Structured vs Unstructured data(3)

Key Differences

Structured data	Unstructured data
Schema already defined	Does not have to follow a schema
Easy to search and analyze	Fast arrival
Can only handle data with a specific schema	Can handle data from any source
Schema changes will cause a lot of troubles	No need to worry about schema changes, as its shifted to the downstream applications that uses this data
Stored in data warehouses	Stored in data lakes



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Query Languages for Data

Pravin Y Pawar

Adapted from Designing Data Intensive Applications
by Martin Kleppmann

Query Languages for Data

- Common ways of querying data
 - Declarative queries
 - Imperative code
- When the relational model was introduced, it included a new way of querying data:
 - SQL is a declarative query language
- whereas IBM IMS and CODASYL queried the database using imperative code

Imperative code vs Declarative queries

- Many commonly-used programming languages are imperative.
- For example, following code retrieves the sharks from the list of animals

```
function getSharks() {  
    var sharks = [];  
    for (var i = 0; i < animals.length; i++) {  
        if (animals[i].family === "Sharks") {  
            sharks.push(animals[i]);  
        }  
    }  
    return sharks;  
}
```

- In the relational algebra, you would instead write: $\text{sharks} = \sigma_{\text{family} = \text{"Sharks"}}(\text{animals})$
where σ is the selection operator, returning only those animals that match the condition family = “Sharks”
- When SQL was defined, it followed the structure of the relational algebra fairly closely:

```
SELECT * FROM animals WHERE family = 'Sharks';
```

Imperative code vs Declarative queries(2)

- In Imperative language
 - instructs the computer to perform certain operations in a certain order
 - stepping through the code, line by line,
 - evaluating conditions
 - updating variables
 - and deciding whether to go around the loop one more time
- In a declarative query language, like SQL or relational algebra,
 - specify the pattern of the data is needed
 - what conditions the results must meet,
 - how you want it to be transformed e.g. sorted, grouped and aggregated
 - **but not how to achieve that goal**
 - Up to the database system's query optimizer to decide
 - which indexes and which join methods to use
 - in which order to execute various parts of the query

Imperative code vs Declarative queries(3)

- A declarative query language is attractive because
 - it is typically **more concise and easier** to work with than an imperative API
- More importantly, it also **hides implementation details** of the database engine
 - makes it possible for the database system to introduce **performance improvements** without requiring **any changes to queries**
- For example, in the imperative code, the list of animals appears in a particular order
 - If the database wants to reclaim unused disk space behind the scenes
 - it might need to move records around, changing the order in which the animals appear
 - Can not be done by the database safely with imperative code
- The SQL doesn't guarantee any particular ordering, and so it doesn't mind if the order changes
 - But if the query is written as imperative code, the database can never be sure whether the code is relying on the ordering or not
 - The fact that SQL is more limited in functionality gives the database much more room for automatic optimizations

Imperative code vs Declarative queries(4)

- Today, CPUs are getting faster by adding more cores
 - not by running at significantly higher clock speeds than before
- Imperative code is very hard to parallelize across multiple cores and multiple machines
 - because it specifies instructions that must be performed in a particular order
- Declarative languages often lend themselves to parallel execution
 - specify only the pattern of the results, but not the algorithm that is used to determine the results
 - database is free to use a parallel implementation of the query language, if appropriate

MapReduce querying

- Programming model for processing large amounts of data in bulk across many machines,
 - popularized by Google
- A limited form of MapReduce is supported by some NoSQL data stores MongoDB and CouchDB
 - as a mechanism for performing read-only queries across many documents
- **MapReduce is neither a declarative query language, nor a fully imperative query API, but somewhere in between!**
 - the logic of the query is expressed with snippets of code, which are called repeatedly by the processing framework
 - based on the map (also known as collect) and reduce (also known as fold or inject) functions that exist in many functional programming languages

MapReduce querying(2)

- For example, imagine you are a marine biologist, and you add an observation record to your database every time you see animals in the ocean
 - Now you want to generate a report saying how many sharks have been sighted per month
- In PostgreSQL you might express that query like this:

```
SELECT date_trunc('month', observation_timestamp) AS observation_month,  
       sum(num_animals) AS total_animals  
  FROM observations  
 WHERE family = 'Sharks'  
 GROUP BY observation_month;
```

- This query
 - first filters the observations to only show species in the Sharks family
 - then groups the observations by the calendar month in which they occurred
 - and finally adds up the number of animals seen in all observations in that month

MapReduce querying(3)

- The same can be expressed with MongoDB's MapReduce feature as follows

```
db.observations.mapReduce(  
  function map() { ②  
    var year = this.observationTimestamp.getFullYear();  
    var month = this.observationTimestamp.getMonth() + 1;  
    emit(year + "-" + month, this.numAnimals); ③  
  },  
  function reduce(key, values) { ④  
    return Array.sum(values); ⑤  
  },  
  {  
    query: { family: "Sharks" }, ①  
    out: "monthlySharkReport" ⑥  
  }  
);
```

- ② The JavaScript function `map` is called once for every document that matches query, with `this` set to the document object.
- ③ The `map` function emits a key (a string consisting of year and month, such as "2013-12" or "2014-1") and a value (the number of animals in that observation).
- ④ The key-value pairs emitted by `map` are grouped by key. For all key-value pairs with the same key (i.e. the same month and year), the `reduce` function is called once.
- ⑤ The `reduce` function adds up the number of animals from all observations in a particular month.
- ⑥ The final output is written to the collection `monthlySharkReport`.

MapReduce querying(4)

- For example, say the observations collection contains these two documents:

```
{  
  observationTimestamp: Date.parse("Mon, 25 Dec 1995 12:34:56 GMT"),  
  family: "Sharks",  
  species: "Carcharodon carcharias",  
  numAnimals: 3  
}  
  
{  
  observationTimestamp: Date.parse("Tue, 12 Dec 1995 16:17:18 GMT"),  
  family: "Sharks",  
  species: "Carcharias taurus",  
  numAnimals: 4  
}
```

- The map function would be called once for each document, resulting in
 - emit("1995-12", 3)
 - emit("1995-12", 4)
- Subsequently, the reduce function would be called with
 - reduce("1995-12", [3, 4]), returning 7.

MapReduce querying(5)

- The map and reduce function are somewhat **restricted** in what they are allowed to do
- Must be pure functions
 - only use the data that is passed to them as input
 - cannot perform additional database queries
 - must not have any side-effects
- **Restrictions allow the database to run the functions anywhere, in any order, and re-run them on failure**
- However, they are nevertheless **powerful**
 - parse strings
 - call library functions
 - perform calculations and more

MapReduce querying(6)

- MapReduce is a fairly low-level programming model for distributed execution on a cluster of machines
 - A usability problem with MapReduce is need to write **two carefully coordinated functions**
 - often **harder** than writing a single query
- Higher-level query languages like SQL can be implemented as a pipeline of MapReduce operations
 - A declarative query language offers more opportunities for a query optimizer to improve the performance of a query
 - MongoDB added support for a declarative query language called aggregation pipeline
- The aggregation pipeline language's expressiveness
 - is similar to a subset of SQL,
 - but it uses a JSON-based syntax rather than SQL's English-sentence
- **The moral of the story is that a NoSQL system may find itself accidentally reinventing SQL, albeit in disguise**

```
db.observations.aggregate([
  { $match: { family: "Sharks" } },
  { $group: {
    _id: {
      year: { $year: "$observationTimestamp" },
      month: { $month: "$observationTimestamp" }
    },
    totalAnimals: { $sum: "$numAnimals" }
  } }
]);
```



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Data Encoding

Pravin Y Pawar

Adapted from Designing Data Intensive Applications
by Martin Kleppmann

Formats for Encoding Data

(at least) two different data representations

- In memory,
 - data is kept in objects, structs, lists, arrays, hash tables, trees etc.
 - data structures are optimized for efficient access and manipulation by the CPU
- In disks, Over networks
 - to write data to a file, or send it over the network, need to encode it
 - as some kind of self-contained sequence of bytes (for example, a JSON document)
 - sequence-of-bytes representation looks quite different from the data structures that are normally used in memory
- Need some kind of translation between the two representations
 - The translation from the in-memory representation to a byte sequence is called encoding
 - aka serialization or marshalling
 - the reverse is called decoding
 - aka parsing, deserialization, unmarshalling

Language-specific formats

Encoding Libraries

- Programming languages come with built-in support for encoding in-memory objects into byte sequences
 - Java has `java.io.Serializable`
 - Ruby has `Marshal`
 - Python has `pickle`
- Encoding libraries are **very convenient!**
 - allow in-memory objects to be saved and restored with minimal additional code

Language-specific formats(2)

Deep Problems

- Integration
 - If tied to a particular programming language, then reading the data in another language is very difficult
 - Store or transmit data in such an encoding, preclude integrating systems with those of other organizations (which may use different languages)
- Decoding
 - To restore data in the same object types, the decoding process needs to instantiate arbitrary classes
 - Frequently a source of security problems
 - if an attacker can get your application to decode an arbitrary byte sequence, they can instantiate arbitrary classes, which in turn often allows them to do terrible things such as remotely executing arbitrary code
- Versioning
 - Is often an afterthought in these libraries
 - Intended for quick and easy encoding of data, they often neglect the inconvenient problems of forward and backward compatibility
- Efficiency
 - CPU time taken to encode or decode, and the size of the encoded structure is also often an afterthought
 - Java's built-in serialization is notorious for its bad performance and bloated encoding
- A bad idea to use language's built-in encoding for anything other than very transient purposes!

Standardized encodings

Textual format

- Standardized encodings can be written and read by many programming languages
 - XML
 - JSON
 - CSV
- Textual formats, and thus somewhat human-readable!
- XML is often criticized for being too verbose and unnecessarily complicated
- JSON's popularity is mainly due to its built-in support by web browsers and simplicity relative to XML
- CSV is albeit less powerful
- Widely known, widely supported, and almost as widely disliked!

Standardized encodings(2)

Subtle problems

- Encoding of numbers
 - Lot of ambiguity around the encoding of numbers
 - In XML and CSV, cannot distinguish between a number and a string that happens to consist of digits
 - JSON distinguishes strings and numbers, but it doesn't distinguish integers and floating-point, and doesn't specify a precision
 - When dealing with large numbers, integers greater than 253 cannot be exactly represented, so such numbers become inaccurate when parsed
 - Workaround is to have two numbers: once as a JSON number and once as a decimal string
- Binary strings
 - JSON and XML have good support for Unicode character strings, i.e. human readable text
 - but they don't support binary strings (sequences of bytes without a character encoding)
 - get around this limitation by encoding the binary data as text using Base64

Standardized encodings(2)

Subtle problems(2)

- Optional schema support
 - Schema support for both XML and JSON is optional
 - Schema languages are quite powerful, and thus quite complicated to learn and implement
 - Use of XML schemas is fairly widespread, but many JSON-based tools don't bother using schemas
 - Applications that don't use XML/JSON schemas need to potentially include additional code to encode/decode data correctly
 - CSV does not have any schema, so it is up to the application to define the meaning of each row and column
 - If an application change adds a new row or column, need to handle that change manually
- Common (data interchange) formats
 - JSON, XML and CSV are good enough for many purposes
 - Likely that they will remain popular, especially as data interchange formats (i.e. sending data from one organization to another)
 - As long as people agree on what the format is, it often doesn't matter how pretty or efficient the format is
 - The difficulty of getting different organizations to agree on anything outweighs most other concerns



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Binary Encoding

Pravin Y Pawar

Adapted from Designing Data Intensive Applications
by Martin Kleppmann

JSON to binary encoding

- For internally used data in organization, less need to use a encoding format
 - For a small dataset, the gains are negligible,
 - For terabytes scale data, the choice of data format can have a big impact
- JSON is less verbose than XML, but both still use a lot of space compared to binary formats
 - led to the development of a profusion of binary encodings for JSON
 - MessagePack, BSON, BJSON, UBJSON, BISON, and Smile etc.
 - none of them are as widely adopted as the textual versions of JSON
- Some of these formats extend the set of datatypes
 - distinguishing integers and floating point, or adding support for binary strings
 - but otherwise they keep the JSON/XML data model unchanged
 - don't prescribe a schema, they need to include all object field names within the encoded data

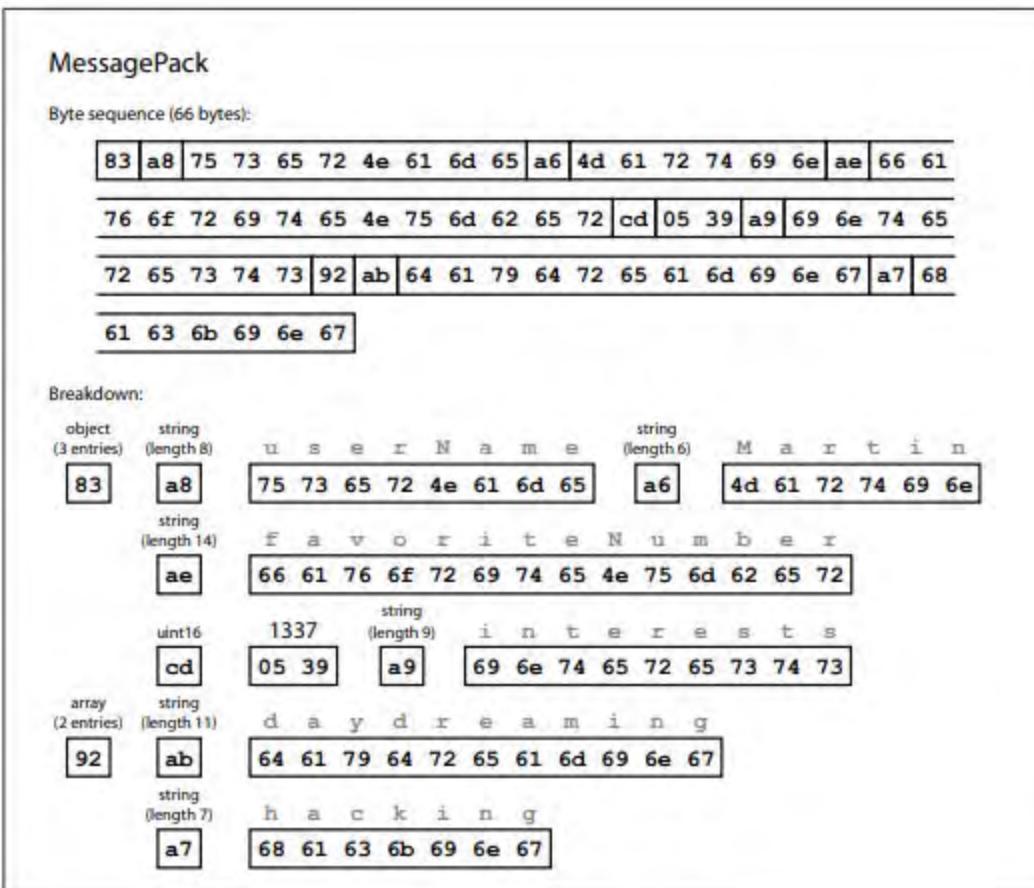
JSON Example Document

Example record

```
{  
    "userNAme": "Martin",  
    "favoriteNAumber": 1337,  
    "interests": ["daydreaming", "hacking"]  
}
```

JSON to MessagePack

Example record encoded using MessagePack



Example record encoded using MessagePack.

JSON to MessagePack(2)

Explanation

- Previous figure shows the byte sequence the JSON document encoding with MessagePack
- The first few bytes are as follows:
 1. The first byte 0x83 indicates that what follows is an object (top four bits = 0x80) with 3 fields (bottom four bits = 0x03)
 2. The second byte 0xa8 indicates that what follows is a string (top four bits = 0xa0) that is 8 bytes long (bottom four bits = 0x08).
 3. The next 8 bytes are the field name `userName` in ASCII. Since the length was indicated previously, there's no need for any marker to tell us where the string ends (or any escaping).
 4. And so on.
- 66 bytes long, which is only a little less than the 81 bytes taken by the textual JSON encoding (with whitespace removed)
 - All the binary encodings of JSON are similar in this regard
 - Not clear whether such a small space reduction(and perhaps a speedup in parsing) is worth the loss of human-readability

Thrift and Protocol Buffers

Better encoding

- Binary encoding libraries that are based on the same principle
 - Require a schema for any data that is encoded
 - Protocol Buffers was originally developed at Google,
 - Thrift was originally developed at Facebook
- The schema in the Thrift interface definition language (IDL)

```
struct Person {  
    1: required string      userName,  
    2: optional i64         favoriteNumber,  
    3: optional list<string> interests  
}
```

- The equivalent schema definition for Protocol Buffers

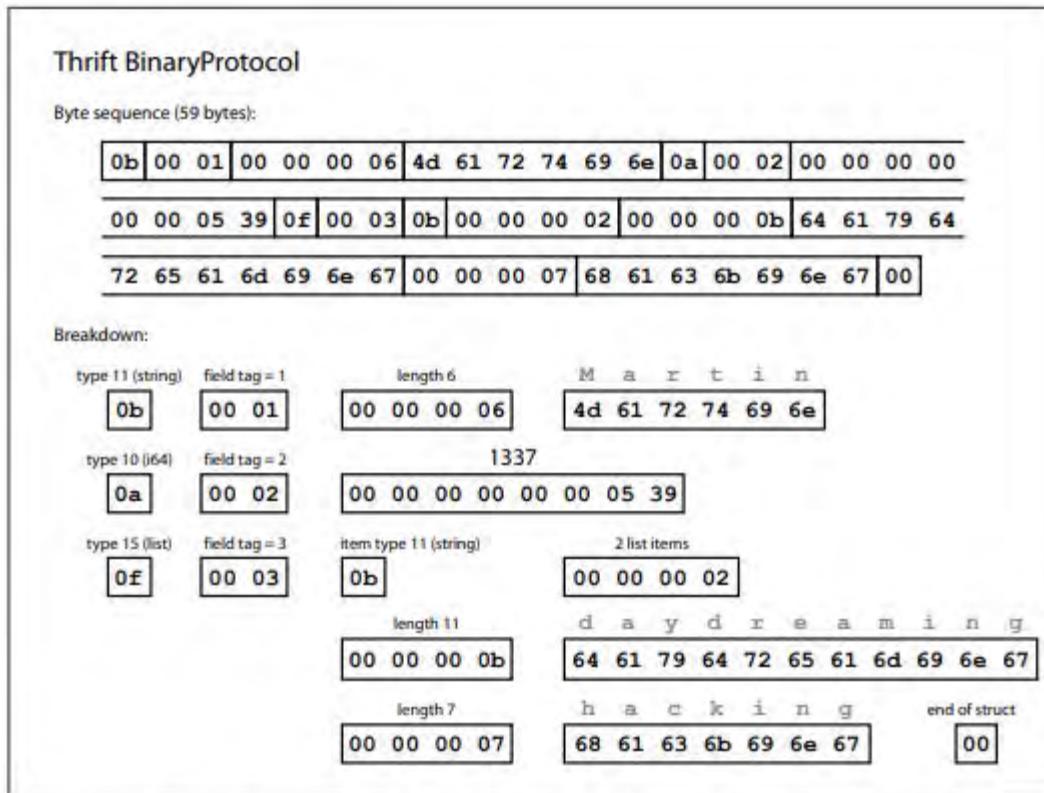
```
message Person {  
    required string user_name      = 1;  
    optional int64  favorite_number = 2;  
    repeated string interests       = 3;  
}
```

- Come with a code generation tool that takes a schema definition and produces classes that implement the schema in various programming languages
 - Application code can call this generated code to encode or decode records of the schema

Thrift

Thrift BinaryProtocol

- Encoding Example in this format takes **59 bytes**



Example record encoded using Thrift's BinaryProtocol.

Thrift(2)

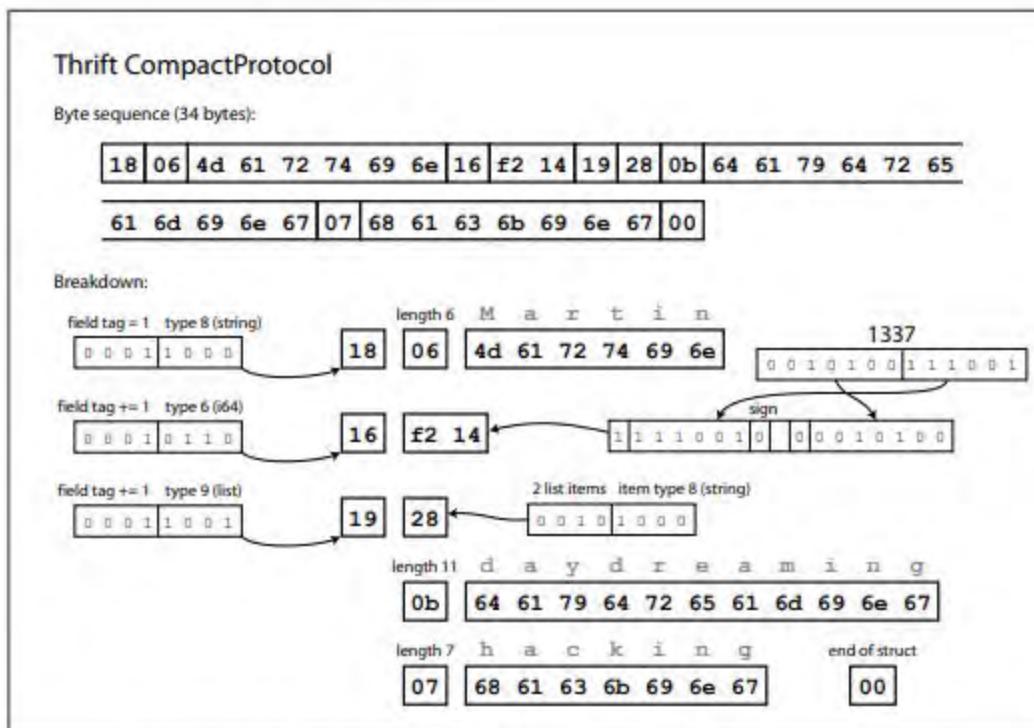
Thrift BinaryProtocol - Explanation

- Each field has a type annotation (a string, integer, list etc.)
- Wherever required, a length indication (length of a string, number of items in a list)
- The strings that appear in the data (“Martin”, “daydreaming”, “hacking”) are also encoded as ASCII (or rather, UTF-8)
- The big difference - no field names (“userName”, “favoriteNumber”, “interests”)
 - Instead, the encoded data contains field tags, which are numbers 1, 2 and 3
 - Those are the numbers that appear in the schema definition
 - Field tags are like aliases for fields
 - A compact way of saying what field we’re talking about, without having to spell out the field name

Thrift(3)

Thrift CompactProtocol

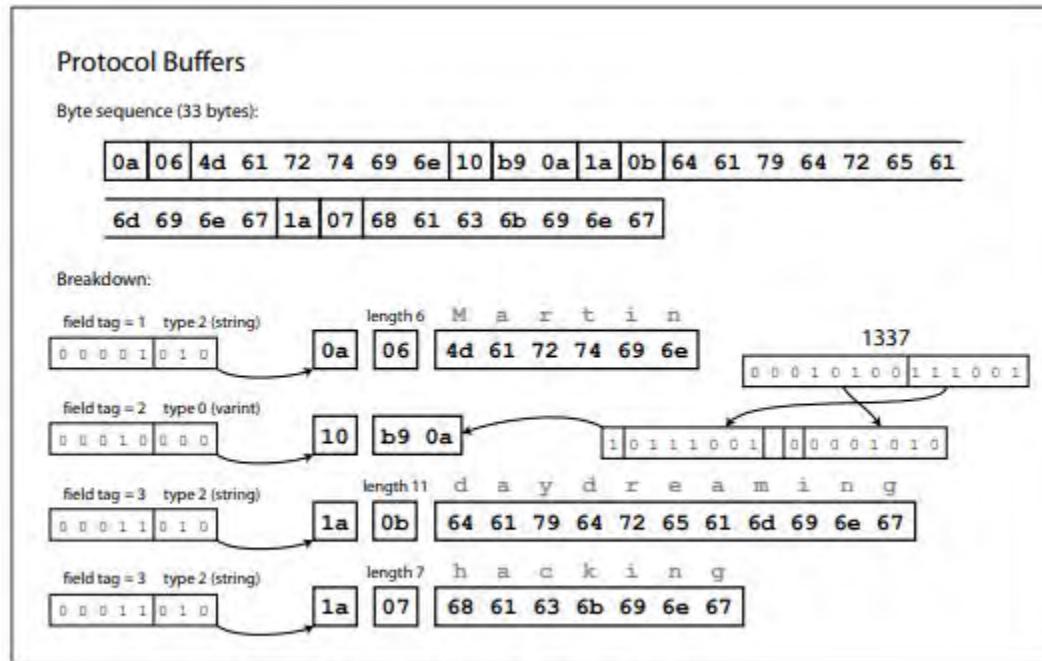
- Packs the same information into only **34 bytes**
 - By packing the field type and tag number into a single byte
 - By using variable-length integers
 - Rather than using a full 8 bytes for the number 1337, it is encoded in two bytes



Example record encoded using Thrift's CompactProtocol.

Protocol Buffers

- Encodes the same data in **33 bytes**
- By bit packing slightly differently, but is otherwise **very similar to Thrift's CompactProtocol**



Example record encoded using Protocol Buffers.

Apache Avro

Binary encoding format different from Protocol Buffers and Thrift

- Started in 2009 as a sub-project of Hadoop, as a result of Thrift not being a good fit for Hadoop's use cases
- Uses a schema to specify the structure of the data being encoded
- Two schema languages:
 - Avro IDL intended for human editing
 - based on JSON that is more easily machine-readable
- Example schema, written in Avro IDL:

```
record Person {  
    string          userName;  
    union { null, long } favoriteNumber = null;  
    array<string>   interests;  
}
```

- The equivalent JSON representation of schema:

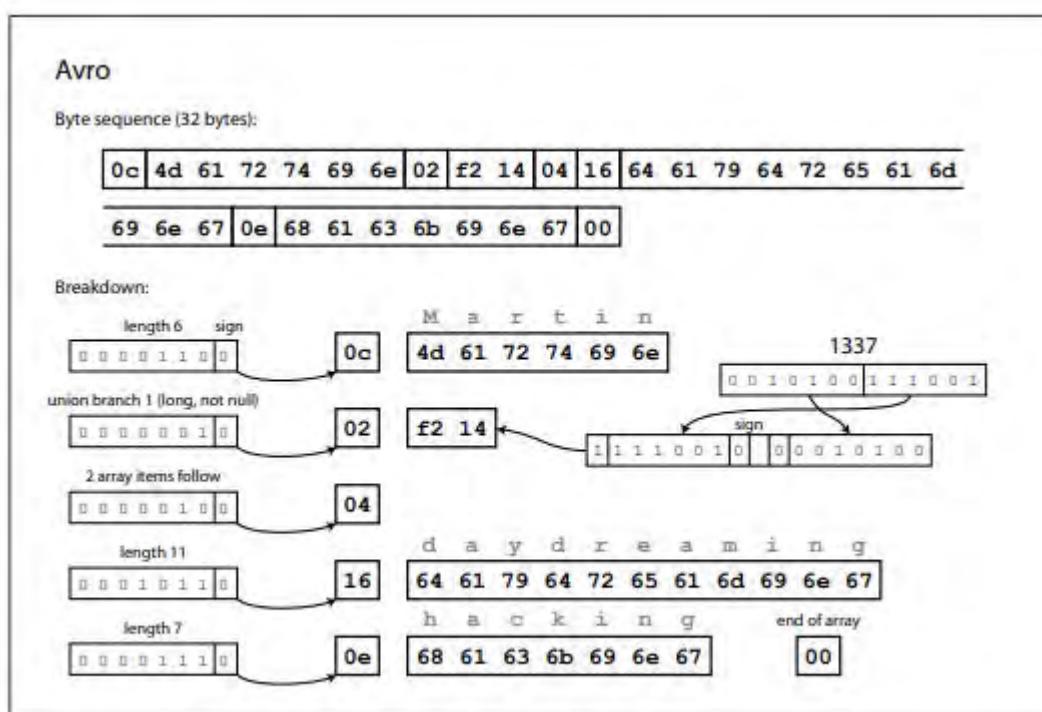
```
{  
    "type": "record",  
    "name": "Person",  
    "fields": [  
        {"name": "userName",      "type": "string"},  
        {"name": "favoriteNumber", "type": ["null", "long"], "default": null},  
        {"name": "interests",     "type": {"type": "array", "items": "string"} }  
    ]  
}
```

No tag numbers in the schema!

Apache Avro(2)

The most compact of all the encodings

- Avro binary encoding is just **32 bytes long**



Example record encoded using Avro.

Apache Avro(3)

Interpretation

- In the byte sequence, nothing to identify fields or their datatypes!
 - The encoding simply consists of values concatenated together
- A string is just a length prefix followed by UTF-8 bytes
 - but there's nothing in the encoded data that tells you that it is a string
 - could just as well be an integer, or something else entirely
- An integer is encoded using a variable-length encoding
- To parse the binary data, need to go through the fields in the order that they appear in the schema and use the schema to tell the datatype of each field
- Means the binary data can only be decoded correctly if the code reading the data is using the exact same schema as the code that wrote the data
 - Any mismatch in the schema between the reader and the writer would mean incorrectly decoded data

Apache Avro(3)

The writer's schema and the reader's schema

- Writer's schema
 - Application wants to encode some data (to write it to a file or database, to send it over the network, etc)
 - it encodes the data using whatever version of the schema that it knows about
 - that schema may be compiled into the application
- Reader's schema
 - Application wants to decode some data (read it from a file or database, receive it from the network, etc.)
 - expecting the data to be in some schema
 - That is the schema the application code is relying on — code may have been generated from that schema during the application's build process
- **The key idea both don't have to be the same — they only need to be compatible!**
 - When data is decoded (read), the Avro library resolves the differences by looking at the writer's schema and the reader's schema side-by-side and translating the data from the writer's schema into the reader's schema

The merits of schemas

- Schema languages are much simpler than XML Schema or JSON Schema
 - Simpler to implement and simpler to use, they have grown to support a fairly wide range of programming languages
- Number of nice properties:
 - Much more compact than the various “binary JSON” variants, since they can omit field names from the encoded data
 - Provides a valuable form of documentation, and because the schema is required for decoding, can be sure that it is up-to-date
 - A database of schemas allows to check forward and backward compatibility of schema changes, before anything is deployed
 - For users of statically typed programming languages, the ability to generate code from the schema is useful, since it enables type-checking at compile time.



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Data Storage Engines and Processing

Pravin Y Pawar

Adapted from Designing Machine Learning Systems
by Chip Huyen

Data Storage Engines and Processing

Two ways

- Data formats and data models specify the interface for how users can store and retrieve data
- Databases (storage engines) are the implementation of how data is stored and retrieved on machines
 - Useful to understand different types of databases as one need to select a database appropriate for application
- Typically, there are two types of workloads that databases are optimized for:
 - transactional processing
 - analytical processing
- Will also explore about
 - ETL / ELT
 - Batch and Stream Processing

Transactional and Analytical Processing

OnLine Transaction Processing (OLTP)

- A transaction refers to the action of buying or selling something
- In the digital world, a transaction refers to **any kind of actions that happen online**:
 - tweeting,
 - ordering a ride through a ridesharing service,
 - uploading a new model,
 - watching a YouTube video, etc.
- Even though these different transactions involve different types of data, the way they're processed is similar across applications!
 - The transactions are inserted as they are generated
 - Occasionally updated when something changes
 - Deleted when they are no longer needed
- This type of processing is known as **OnLine Transaction Processing (OLTP)**.
- These transactions often involve users
 - They need to be processed fast (low latency) so that they don't keep users waiting
 - The processing method needs to have high availability — e.g. the processing system needs to be available any time a user wants to make a transaction
 - If system can't process a transaction, that transaction won't go through.

Transactional and Analytical Processing(2)

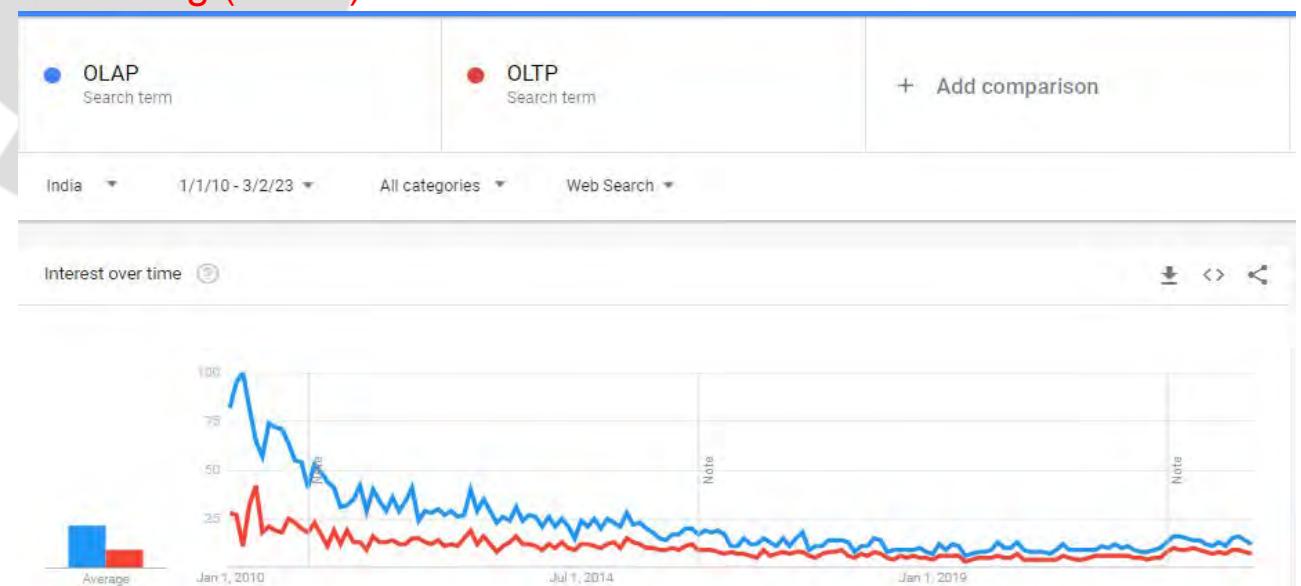
ACID and BaSE Properties of Transaction

- Transactional databases are designed to process online transactions and satisfy the **low latency, high availability**
 - Usually think of **ACID** (**A**tomicity, **C**onsistency, **I**solation, **D**urability)
- Atomicity
 - Guarantee that all the steps in a transaction are completed successfully as a group
 - If any step between the transaction fails, all other steps must fail also
 - If a user's payment fails, you don't want to still assign a driver to that user.
- Consistency
 - Guarantee that all the transactions coming through must follow predefined rules
 - A transaction must be made by a valid user
- Isolation
 - Guarantee that two transactions happen at the same time as if they were isolated
 - Two users accessing the same data won't change it at the same time
- Durability
 - Guarantee that once a transaction has been committed, it will remain committed even in the case of a system failure
 - User ordered a ride and his phone dies, he still want his ride to come

Transactional and Analytical Processing(3)

OnLine Analytical Processing (OLAP)

- Transactional databases are often **row-major**
 - Each transaction is often processed as a unit separately from other transactions
 - Might not be efficient for questions such as “What’s the average price for all the rides in September in San Francisco?”
- Analytical databases
 - Such kind of analytical question requires **aggregating data in columns across multiple rows of data**
 - Efficient with queries that allow you to look at data from different viewpoints
 - This type of processing **OnLine Analytical Processing (OLAP)**!
- However, both the terms OLTP and OLAP have become outdated



Transactional and Analytical Processing(4)

OLTP and OLAP have become outdated - reasons

- First, the separation of transactional and analytical databases was due to limitations of technology
 - Was hard to have databases that could handle both transactional and analytical queries efficiently
 - Today,
 - Transactional databases can handle analytical queries, such as CockroachDB
 - Analytical databases can handle transactional queries, such as Apache Iceberg
- Second, in the traditional OLTP or OLAP paradigms, storage and processing are tightly coupled
 - how data is stored is also how data is processed
 - Result in the same data being stored in multiple databases and use different processing engines to solve different types of queries
- An interesting paradigm in the last decade has been to decouple storage from processing (also known as compute)
 - the data can be stored in the same place, with a processing layer on top that can be optimized for different types of queries
 - adopted by many data vendors including Google's BigQuery, Snowflake, IBM, and Teradata

Transactional and Analytical Processing(5)

OLTP and OLAP have become outdated - reasons

- Third, “online” has become an overloaded term that can mean many different things
 - initially meant “connected to the Internet”
 - grew to also mean “in production” — a feature is online after that feature has been deployed in production
 - might refer to the speed at which data is processed and made available: **online, nearline, or offline**
 - Online processing means data is immediately available for input/output
 - Nearline, which is short for near-online, means data is not immediately available, but can be made online quickly without human intervention
 - Offline means data is not immediately available, and requires some human intervention to become online
- As the speed at which applications respond to users queries has become a competitive advantage, it's become more and more important to make data available for use as fast as possible.
 - In many use cases, companies want online processing not just for transactional queries but also for analytical queries. Online, in this case, is synonymous to **“real-time”**.

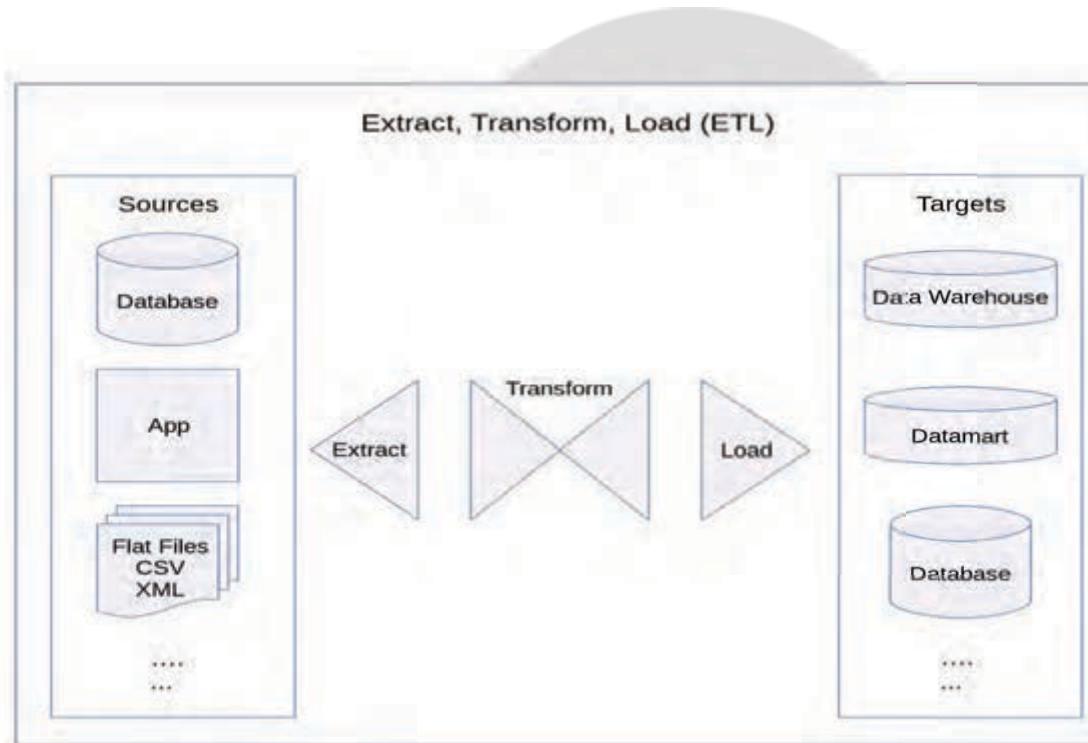
ETL: Extract, Transform, Load

- ETL (extract, transform, load) was all the rage in the data world
 - still relevant today for ML applications
 - refers to the general purpose processing and aggregating data into the shape and the format that you want
- Extract
 - Extracting the data required from data sources
 - Data will likely come from multiple sources in different formats - may be corrupted or malformatted
 - Need to validate data and reject the data that doesn't meet requirements
 - Doing it correctly can save you a lot of time downstream
- Transform
 - Meaty part of the process, where most of the data processing is done
 - Might want to join data from multiple sources and clean it
 - Might want to standardize the value ranges
 - Might apply operations such as transposing, deduplication, sorting, aggregating, deriving new features, more data validating, etc.
- Load
 - Deciding how and how often to load transformed data into the target destination
 - which can be a file, a database, or a data warehouse

ETL: Extract, Transform, Load(2)

An overview of the ETL process

- The idea of ETL sounds simple but powerful, and it's the underlying structure of the data layer at many organizations



ETL: Extract, Transform, Load(3)

ETL to ELT

- Because of ubiquitous Internet and powerful hardware, collecting data suddenly became so much easier!
 - The amount of data grew rapidly
 - The nature of data also changed
 - The number of data sources expanded
 - The data schemas evolved
- ELT
 - Finding it difficult to keep data structured, some companies had this idea:
 - “Why not just store all data in a data lake so we don’t have to deal with schema changes? Whichever application needs data can just pull out raw data from there and process it.”
 - This process of loading data into storage first then processing it later is sometimes called ELT (extract, load, transform)
 - This paradigm allows for the fast arrival of data since there’s little processing needed before data is stored
- As data keeps on growing, this idea becomes less attractive
 - expensive to store everything, and inefficient to search through a massive amount of raw data for the piece of data that you want
 - companies switch to running applications on the cloud and infrastructures become standardized, data structures also become standardized
 - committing data to a predefined schema becomes more feasible

Batch vs Stream Processing

Historical vs Streaming data

- Once data arrives in databases, data lakes or data warehouses its **historical data**
 - **Batch jobs** – kicked off periodically are used to process historical data
 - **Batch processing** is processing where data is processed in batches of jobs
 - e.g. once a day, compute average surge charge for rides in the last day
 - **Hadoop MapReduce and Spark** are know batch processing frameworks
- Data that is still streaming in is **streaming data**
 - Ingested through real-time transports like **Apache Kafka and Amazon Kinesis**
 - **Stream processing** refers to doing computation on streaming data
 - Computations can be kicked off **periodically** but periods are much shorter than periods of batch jobs
 - Computation can be kicked off **whenever need arises**
 - **Reduces latency** as data can be processed as soon as its generated w/o writing to database
 - E.g. whenever a user requests a ride, process data stream to see what drivers are currently nearby
 - **Apache Flink, Storm, Spark Streaming** provides these processing capabilities

Batch vs Stream Processing(2)

Static vs Dynamic features

- In ML , batch processing is used to compute feature that changes less often
 - e.g. drivers rating – does change quite often
 - **Batch features** – extracted through batch processing are **static features!**
- Stream processing is used to compute features that changes quickly
 - For example,
 - how many drivers are available right now
 - how many rides have been requested in last one minute
 - how many rides will be finished in next two minutes
 - Median price of last 10 rides in a particular area
 - Features reflecting current state of system are important for price estimations
 - **Streaming features** – extracted through stream processing are **dynamic features!**

Batch vs Stream Processing(3)

Stream Processing Engines

- Stream computation engines are needed to do computation on data streams
 - MapReduce and Spark are batch processing engines
 - Simple streaming computations might be carried out with built-in capabilities of real-time transports like Apache Kafka i..e Kafka Stream processing
- For ML systems that uses dynamic features, computations are rarely simple!
 - For example, fraud detection can have hundreds of features to be considered
 - Stream feature extraction logic can require complex queries with join and aggregation along different dimensions
 - Requires efficient stream processing engines such as Apache Flink, KSQL and Spark Streaming
 - Nice SQL abstraction over streaming data is provided by Flink and KSQL
- Stream processing is more difficult because data amount is unbounded and data comes in at variable rates and speeds
 - Easier to make a stream processor do batch processing than to make batch processor to do stream processing
 - Flink's maintainers arguing for years – Batch processing is a special case of stream processing



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Modes of Data Flow

Pravin Y Pawar

Adapted from Designing Data Intensive Applications
by Martin Kleppmann

Modes of Data Flow

- Many ways how data can flow from one process to another
 - Who encodes the data, and who decodes it?
- Most common ways how data flows between processes:
 - Via Databases, where the process writing to the database encodes the data, and the process reading from the database decodes it.
 - Via RPC and REST APIs, where the client encodes a request, the server decodes the request and encodes a response, and the client finally decodes the response.
 - Via Asynchronous message-passing (using message brokers or actors), where nodes communicate by sending each other messages that are encoded by the sender and decoded by the recipient

Data flow through databases

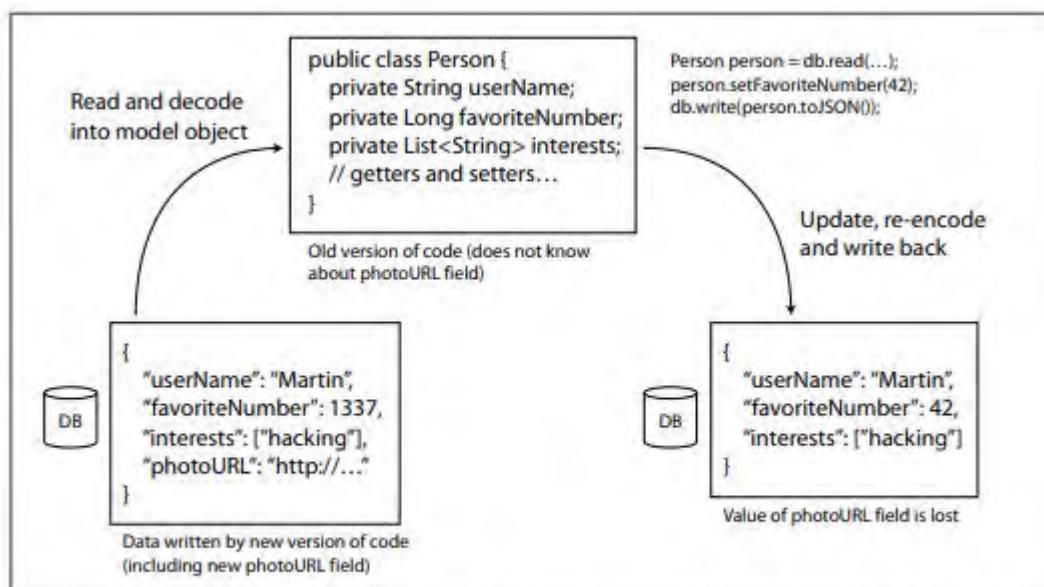
Data Compatibility

- In a database, the process that writes to the database encodes the data, and the process that reads from the database decodes it
- May just be a single process accessing the database, in which case the reader is simply a later version of the same process
 - Backward compatibility is clearly necessary here, otherwise won't be able to decode what was previously wrote
- Common for several different processes to be accessing a database at the same time
 - For example, several different applications or services, or several instances of the same service
 - Either way, in an environment where the application is changing,
 - Likely that some processes accessing the database will be running newer code and some will be running older code
 - A new version is currently being deployed in a rolling upgrade, so some instances have been updated
 - Others haven't yet!
 - Means that a value in the database may be written by a newer version of the code, and subsequently read by an older version of the code that is still running
 - Forward compatibility is also often required for databases

Data flow through databases(2)

Additional snag

- If a field is added to a record schema, and the newer code writes a value for that new field to the database. Subsequently, an older version of the code (which doesn't yet know about the new field) reads the record, updates it, and writes it back
- In this situation, the desirable behavior is usually for the old code to keep the new field intact, even though it couldn't be interpreted
 - The encoding formats support such preservation of unknown fields, but sometimes need to take care at an application level,



When an older version of the application updates data previously written by a newer version of the application, data may be lost if you're not careful.

Data flow through databases(3)

Different values written at different times

- A database generally allows any value to be updated at any time
 - Means within a single database may have some values that were written five milliseconds ago, and some values that were written five years ago
- Data outlives code
 - When deploy a new version of your application, may entirely replace the old version with the new version within a few minutes
 - Same is not true of database contents: the five-year-old data will still be there
- Schema Evolution
 - Rewriting (migrating) data to a new schema is certainly possible, but it's an expensive thing to do on a large dataset, so mostly avoided
 - Simple schema changes, such as adding a new column with a null default value, without rewriting existing data
 - Schema evolution thus allows the entire database to appear as if it was encoded with a single schema, even though the underlying storage may contain records encoded with various historical versions of the schema

Data flow through databases(4)

Archival storage

- Snapshot of database can be taken from time to time, say for backup purposes or for loading into a data warehouse
 - Data dump will typically be encoded using the latest schema, even if the original encoding in the source database contained a mixture of schema versions from different eras
 - Copying the data anyway, might as well encode the copy of the data consistently
- As the data dump is written in one go, and is thereafter immutable, formats like Avro object container files are a good fit
 - Good opportunity to encode the data in an analytics-friendly column-oriented format such as Parquet

Data flow through services

Client and Server

- Processes that need to communicate over a network, there are a few different ways of arranging that communication
 - The most common arrangement is to have two roles: clients and servers
 - The servers expose an API over the network, and the clients can connect to the servers to make requests to that API
 - The API exposed by the server is known as a service
- Web Clients
 - Clients (web browsers) make requests to web servers, making GET requests to download HTML, CSS, JavaScript, images etc. or making POST requests to submit data to the server
 - The API consists of a standardized set of protocols and data formats (HTTP, URLs, SSL/TLS, HTML, etc.)
 - Because web browsers, web servers and website authors mostly agree on these standards, can use any web browser to access any website
- Other Clients
 - A native app running on a mobile device or a desktop computer can also make network requests to a server
 - A client-side JavaScript application running inside a web browser can use XMLHttpRequest to become a HTTP client (aka Ajax)
 - The server's response is typically not HTML for displaying to a human, but rather data in an encoding that is convenient for further processing by the clientside
 - application code (such as JSON)
 - API is implemented on top of HTTP is application-specific,
 - Client and server need to agree on the details of that API

Data flow through services(2)

Service-oriented architecture (SOA) / Microservices Architecture

- Service-oriented architecture (SOA)
 - Server can itself be a client to another service
 - Often used to decompose a large application into smaller services by area of functionality, such that one service makes a request to another when it requires some functionality or data from that other service
 - more recently refined and rebranded as microservices architecture!
- A key design goal is to make the application easier to change and maintain by making services independently deployable and evolvable
 - For example, each service should be owned by one team, and that team should be able to release new versions of the service frequently without having to coordinate with other teams
- Should expect old and new versions of servers and clients to be running at the same time
 - Data encoding used by servers and clients must be compatible across versions of the service API

Data flow through services(3)

Web services

- HTTP is used as the underlying protocol for talking to the service
 - Two popular approaches: REST and SOAP
- REST is not a protocol, but rather a design philosophy that builds upon the principles of HTTP
 - emphasizes simple data formats, using URLs for identifying resources, and using HTTP features for cache control, authentication, and content type negotiation
 - gaining popularity compared to SOAP, at least in the context of cross-organizational service integration and is often associated with microservices
 - An API designed according to the principles of REST is called RESTful
- SOAP is an XML-based protocol for making network API requests
 - Most commonly used over HTTP, it aims to be independent from HTTP and avoids using most HTTP features
 - Instead comes with a sprawling and complex multitude of related standards (the web service framework, known as WS-*)
 - The API of a SOAP web service is described using an XML-based language called WSDL
 - As WSDL is not designed to be human-readable, and as SOAP messages are often too complex to construct manually, users of SOAP rely heavily on tool support, code generation and IDEs
 - Although SOAP is still used in many large enterprises, it has fallen out of favor in most smaller companies.

Data flow through services(4)

Remote procedure calls (RPC)

- Web services are merely the latest incarnation of a long lineage of technologies for making API requests over a network, many of which received a lot of hype but have serious problems
 - Enterprise JavaBeans (EJB) and Java Remote Method Invocation (RMI) are limited to Java
 - DCOM is limited to Microsoft platforms
 - CORBA is excessively complex, and does not provide backward or forward compatibility
- All of these are based on the idea of a Remote Procedure Call (RPC), which has been around since the 1970s
 - RPC tries to make a request to a remote network service look the same as calling a function or method in programming language, within the same process (aka **location transparency**)
 - Although this seems convenient at first, the approach is fundamentally flawed
 - No point trying to make a remote service look too much like a local object in programming language, because it's a fundamentally different thing

Data flow through services(5)

Current directions for RPC

- **RPC isn't going away!**
 - Various RPC frameworks have been built on top of all the encodings:
 - Thrift and Avro come with RPC support included
 - gRPC is a RPC implementation using Protocol Buffers
- New generation of RPC frameworks is more explicit about the fact that a remote request is different from a local function call
 - Also provide service discovery — that is, allowing a client to find out at which IP address and port number it can find a particular service
 - Custom RPC protocols with a binary encoding format can achieve better performance than something generic like JSON over REST
- RESTful API has other significant advantages
 - good for experimentation and debugging (can simply make requests to it using a web browser or the command-line tool curl)
 - supported by all main stream programming languages and platforms
 - a vast ecosystem of tools(servers, caches, load balancers, proxies, firewalls, monitoring, debugging tools, testing tools, etc.)
- **For these reasons, REST seems to be the predominant style for public APIs.**
 - Focus of RPC frameworks is on requests between services owned by the same organization, typically within the same datacenter

Message passing data flow

Asynchronous message-passing systems

- Common ways of data exchange
 - REST and RPC (where one process sends a request over the network to another process, and expects a response as quickly as possible),
 - Databases (where one process writes encoded data, and another process reads it again sometime in the future)
- Asynchronous message-passing systems, which are **somewhere between RPC and databases**
 - similar to RPC in that a client's request (usually called a message) is delivered to another process with low latency
 - similar to databases in that the message is not sent via a direct network connection, but goes via an intermediary called a message broker which stores the message temporarily
 - also called a message queue or message-oriented middleware

Message passing data flow(2)

Message broker's Advantages compared to direct RPC

- can act as a buffer if the recipient is unavailable or overloaded, and thus improve system reliability
- can automatically redeliver messages to a process that crashed, and thus prevent messages from being lost
- avoids the sender needing to know the IP address and port number of the recipient
- allows one message to be sent to several recipients
- logically decouples the sender from the recipient (the sender just publishes messages and doesn't care who consumes them)
- **Message-passing communication is usually one-way**
 - sender normally doesn't expect to receive a reply to its messages
 - possible for a process to send a response, but this would usually be done on a separate channel
- This is what makes it **asynchronous**: the sender doesn't wait for the message to be delivered, but simply sends it and then forgets about it.

Message passing data flow(3)

Message brokers

- Landscape
 - In past, dominated by commercial enterprise software by companies such as TIBCO, IBM WebSphere, and webMethods.
 - More recently, open source implementations such as RabbitMQ, ActiveMQ, HornetQ, NATS and Apache Kafka have become popular
- In general, message brokers are used as follows:
 - one process sends a message to a named queue or topic
 - broker ensures that the message is delivered to one or more consumers or subscribers of that queue or topic
 - can be many producers and many consumers on the same topic
- A topic provides only one-way data flow!
- Message brokers typically don't enforce any particular data model
 - message is just a sequence of bytes, with some metadata, so can use any encoding format
 - If the encoding is backward and forward compatible, have the greatest flexibility to change publishers and consumers independently, and deploy them in any order



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Types of Data Architecture - I

Pravin Y Pawar

Adapted from Fundamentals of Data Engineering
by Joe Reis and Matt Housley

Data Warehouse

- Is a central data hub used for reporting and analysis
 - Data is typically highly formatted and structured for analytics use cases
 - Among oldest and most well-established data architectures
- Per Bill Inmon, father of data warehouse, it is
 - A subject oriented, integrated, non-volatile and time-variant collection of data in support of management's decisions
- In past, widely used at enterprises with significant budgets – expensive and labour-intensive
- Since then, scalable, pay-as-you-go model has made it accessible even to tiny companies

Data Warehouse(2)

Organizational data warehouse architecture

- Organizes data associated with business team structures and processes
 - Separates online analytical processing (OLAP) from production databases (OLTP)
 - Centralizes and organizes data
- ETL (Extract, Transfer, Load)
 - DW pulls data from application systems using ETL
 - Extract phase pull data from source systems
 - Transformation phase cleaned and standardizes data, applies business logic
 - Load phase pushes data into DW target database system
 - Data is then loaded into multiple data marts that serve analytical needs for specific lines of business

Data Warehouse(3)

ELT DW Architecture

- ELT – Extract, Load and Transfer
- Data gets moved more or less directly from production systems into a staging area in the data warehouse
 - Staging stage indicates that data is in raw form
 - Transformations are directly handled by data warehouse system
 - Intention is to take advantage of massive computational power of cloud data processing tools
 - Data is processed in batches and transformed output is written into tables and views for analytics
- Transformation-on-Read ELT
 - Popularized during big data growth in Hadoop ecosystem

Data Warehouse(4)

Cloud data warehouse

- Represents evolution of on-premises data warehouse architecture and have led to significant changes to organizational architecture
 - Amazon Redshift kicked off cloud data warehouse revolution
 - Google BigQuery, Snowflake popularized the approach
 - Can spin up a Redshift cluster on demand, scaling it up over time as data and analytics grows
 - Can spin up cluster to serve specific workloads and quickly delete clusters when not needed
- Separate compute from storage
 - Data is stored in object storage allowing virtually limitless storage
 - Gives option to spin up computing power on demand providing ad hoc big data capabilities without long-term cost of thousands of nodes
- As cloud data warehouse matures, line between data warehouse and date lake will continue to blur!

Data Warehouse(5)

Data Marts

- More refined subset of a warehouse designed to serve analytics and reporting, focused on a single sub organization, department or line of business
 - Every department has its own data mart, specific to its needs
 - Contrast to full data warehouse that serves the broader organization or business
- Exists for two reasons
 - Makes data more easily accessible to analysts and report developers
 - Provide additional stage of transformation beyond that provided by initial ETL/ELT pipelines
- Significantly improves performance if reports or analytics requires complex joins and aggregations of data, especially when raw data is large
 - Transform processes can populate data mart with joined and aggregated data to improve performance of live queries

Data Lake

- Most popular architectures that appeared during big data era
 - Instead of imposing tight structural limitations on data, simply dump all data – structured and unstructured – into central location
- Data lake 1.0
 - Started with HDFS , moved to cloud based object storage – extremely limitless and cheap storage costs
 - Allows immense amount of data of any size and type to be stored
 - When needs to be queried or transformed, have access to nearly unlimited computing power by spinning cluster on demand with favourite tools – MapReduce, Spark, Presto, Hive
- Had serious shortcomings – dumping ground – data swamps, dark data and WORN
 - Data grew to unmanageable sizes, with little in way of schema management, data cataloguing and discovery tools
 - Write-only – updates and deletes become headaches
 - Processing also challenge – joins were nightmares!
- Many organizations found significant value in data lakes – Netflix and Facebook
 - Had resources to build successful data practices and create their Hadoop based tools

Data lakehouse

- Various players sought to enhance limitations of first generations data lakes
 - Databricks introduced data lakehouse – convergence between data warehouse and data lakes
 - Incorporates the controls, data management and data structures found in data warehouse while still housing data in object storage and supporting variety of query and transformation engines
 - Supports Atomicity, consistency, durability and isolation (ACID)
- Technical architecture of cloud data warehouses have evolved to be very similar to data lake architecture
 - Separates compute from storage
 - Support petabyte-scale queries
 - Store a variety of unstructured and semi-structured objects
 - Integrate with advanced processing technologies such as Spark and Beam
- Convergence will continue but still will exist as different architectures
 - Vendors offering data platforms that combine data lake and data warehouse capabilities
 - AWS, Azure, Google Cloud and Snowflake, Databricks are class leaders
 - Offering tightly integrated tools for working with data, running the gamut from relational to completely unstructured



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Types of Data Architecture - II

Pravin Y Pawar

Adapted from Fundamentals of Data Engineering
by Joe Reis and Matt Housley

Modern Data Stack

- Currently trendy analytics architecture
 - Core aim is to reduce complexity and increase modularization
- Past data stacks relied on expensive, monolithic toolsets
- MDS uses cloud-based, plug-and-play, easy-to-use, off-the-shelf components to create a modular and cost-effective data architecture
 - Data Pipelines
 - Cloud Storage
 - Transformation
 - Data Management / Governance
 - Monitoring, Visualization and exploration
- Still in flux, tools are changing and evolving rapidly
 - Key concept of plug-and-play modularity with easy-to-understand pricing and implementation is way of future
 - In analytics engineering, MDS is and will continue to be default choice of data architecture!

Lambda Architecture

- Old days (early to mid-2010s) popularity of working with streaming data exploded
 - Kafka as highly scalable message queue and f/w Apache Storm and Samza for analytics
 - Allowed to perform new types of analytics and modeling on large amounts of data
 - Needed to figure out how to reconcile batch and streaming data in single architecture
 - Answer: Lambda architecture!
- Three systems : **batch, streaming and serving** are operating independently
 - Source is immutable and append-only, sending data to two destinations : stream and batch
 - In-stream processing serves data with lowest possible latency in speed layer, usually NoSQL DB
 - Batch layer process and transforms data in data warehouse, creating precomputed and aggregated views of data
 - Serving layer provides a combined view by aggregating query results from two layers
- Managing multiple systems with different codebases is difficult, creating error-prone systems with code and data that are extremely difficult to reconcile!

Kappa Architecture

- As a response to shortcomings of Lambda architecture, Jay Kreps proposed an alternative
 - Just use stream processing platform as backbone for all data handling
 - Ingestion, storage and serving – facilitating a true event based architecture
 - Real-time and batch processing can be applied seamlessly to same data
 - By reading live event stream directly and replaying large chunks of data for batch processing
- Not yet widely adopted!
 - Streaming itself is still a bit of mystery to many companies – easy to talk, harder to execute
 - Turns out to be complicated and expensive in practice
 - Batch storage and processing remain much more efficient and cost-effective for enormous historical data

Dataflow model

Unified batch and streaming

- Lambda & Kappa both provided inspiration and groundwork for unification of batch and stream
 - Trying to tie together complicated tools that were not natural fits in the first place
 - Central problem is managing batch and stream processing with multiple code paths
 - Kappa relies on unified queuing and storage layer, still depends on different tools
- Google attempted to solve it by Dataflow model and Apache Beam as its implementation
 - Core idea is to view all data as events, aggregation is performed over various types of windows
 - Ongoing real-time event streams are unbounded data
 - Data batches are simply bounded event streams and boundaries provides natural window
 - Windows can tumbling , sliding
 - Real-time and batch processing happens in same system using nearly identical code
- Philosophy “batch as a special case of streaming” is now more pervasive
 - Frameworks such as Flink and Spark have adopted a similar approach!

Data Mesh

- Recent response to sprawling monolithic data platforms, such as centralized data lakes and warehouse and great divide of data – operational and analytical data
 - Attempts to invert the challenges of centralized data architecture – by applying concepts of domain driven design on data architecture
- Big part of data mesh is **decentralization**,
- Per Zhamak Dehghani,
 - In order to decentralize the monolithic data platforms, we need to reverse how we think about data, its locality and ownership. Instead of flowing the data from domains into a centrally owned data lake or platform, domain need to host and serve their domain datasets in easily consumable way.
- Four key components
 - Domain-oriented decentralized data ownership and architecture
 - Data as a product
 - Self-serve data infrastructure as a platform
 - Federated computational governance



Thank You!

In our next session:



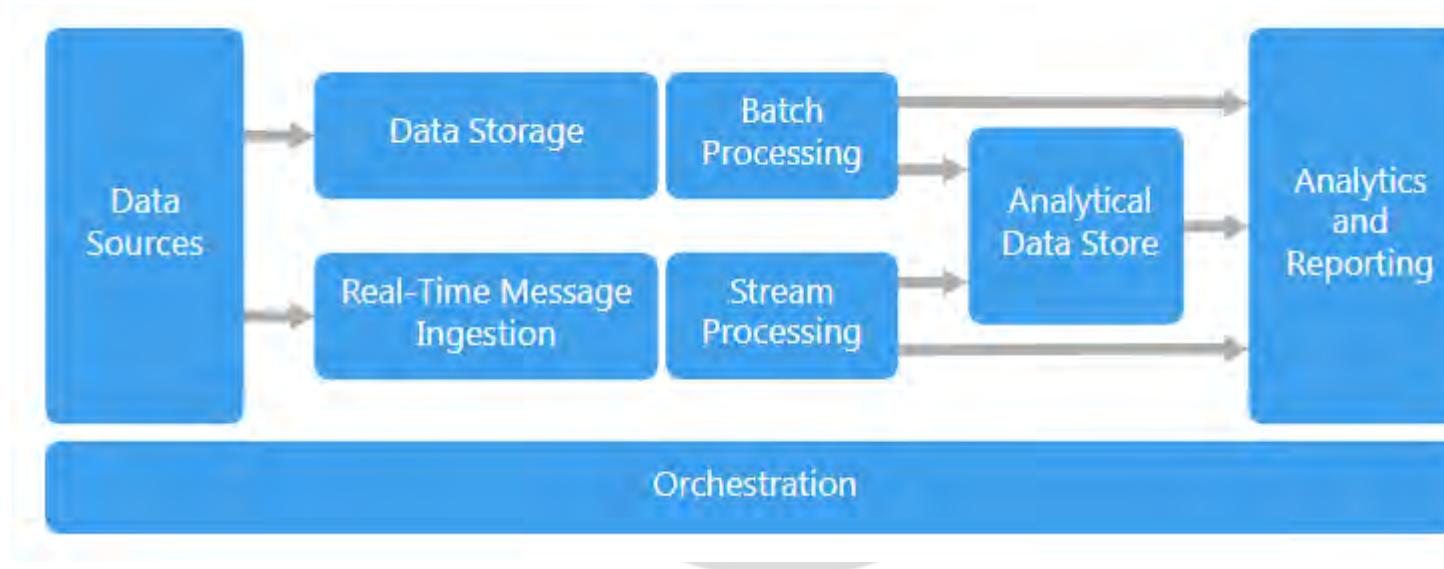
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Generalized Architecture of Big Data Systems

Pravin Y Pawar

Big data architecture style

- is designed to handle the ingestion, processing, and analysis of data that is too large or complex for traditional database systems.



Source : <https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/big-data>

Big Data Applications

Workloads

- Big data solutions typically involve one or more of the following types of workload:
 - ✓ Batch processing of big data sources at rest
 - ✓ Real-time processing of big data in motion
 - ✓ Interactive exploration of big data
 - ✓ Predictive analytics and machine learning

Big Data Systems Components

Components

- Most big data architectures include some or all of the following components:
 - ✓ Data sources
 - All big data solutions start with one or more data sources like databases, files, IoT devices etc
 - ✓ Data Storage
 - Data for batch processing operations is typically stored in a distributed file store that can hold high volumes of large files in various formats.
 - ✓ Batch processing
 - Because the data sets are so large, often a big data solution must process data files using long-running batch jobs to filter, aggregate, and otherwise prepare the data for analysis. Usually these jobs involve reading source files, processing them, and writing the output to new files.
 - ✓ Real-time message ingestion
 - If the solution includes real-time sources, the architecture must include a way to capture and store real-time messages for stream processing.
 - ✓ Stream processing
 - After capturing real-time messages, the solution must process them by filtering, aggregating, and otherwise preparing the data for analysis. The processed stream data is then written to an output sink.
 - ✓ Analytical data store
 - Many big data solutions prepare data for analysis and then serve the processed data in a structured format that can be queried using analytical tools. The analytical data store used to serve these queries can be a Kimball-style relational data warehouse, as seen in most traditional business intelligence (BI) solutions.
 - ✓ Analysis and reporting
 - The goal of most big data solutions is to provide insights into the data through analysis and reporting.
 - ✓ Orchestration
 - Most big data solutions consist of repeated data processing operations, encapsulated in workflows, that transform source data, move data between multiple sources and sinks, load the processed data into an analytical data store, or push the results straight to a report or dashboard. To automate these workflows, you can use an orchestration technology such Azure Data Factory or Apache Oozie and Sqoop.

Big data architecture Usage

When to use this architecture

- Consider this architecture style when you need to:
 - ✓ Store and process data in volumes too large for a traditional database
 - ✓ Transform unstructured data for analysis and reporting
 - ✓ Capture, process, and analyze unbounded streams of data in real time, or with low latency

Big data architecture Benefits

Advantages

- Technology choices
 - ✓ Variety of technology options in open source and from vendors are available
- Performance through parallelism
 - ✓ Big data solutions take advantage of parallelism, enabling high-performance solutions that scale to large volumes of data.
- Elastic scale
 - ✓ All of the components in the big data architecture support scale-out provisioning, so that you can adjust your solution to small or large workloads, and pay only for the resources that you use.
- Interoperability with existing solutions
 - ✓ The components of the big data architecture are also used for IoT processing and enterprise BI solutions, enabling you to create an integrated solution across data workloads.

Big data architecture Challenges

Things to ponder upon

- Complexity
 - ✓ Big data solutions can be extremely complex, with numerous components to handle data ingestion from multiple data sources. It can be challenging to build, test, and troubleshoot big data processes.
- Skillset
 - ✓ Many big data technologies are highly specialized, and use frameworks and languages that are not typical of more general application architectures. On the other hand, big data technologies are evolving new APIs that build on more established languages.
- Technology maturity
 - Many of the technologies used in big data are evolving. While core Hadoop technologies such as Hive and Pig have stabilized, emerging technologies such as Spark introduce extensive changes and enhancements with each new release.



Thank You!

In our next session: Streaming Data Systems



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

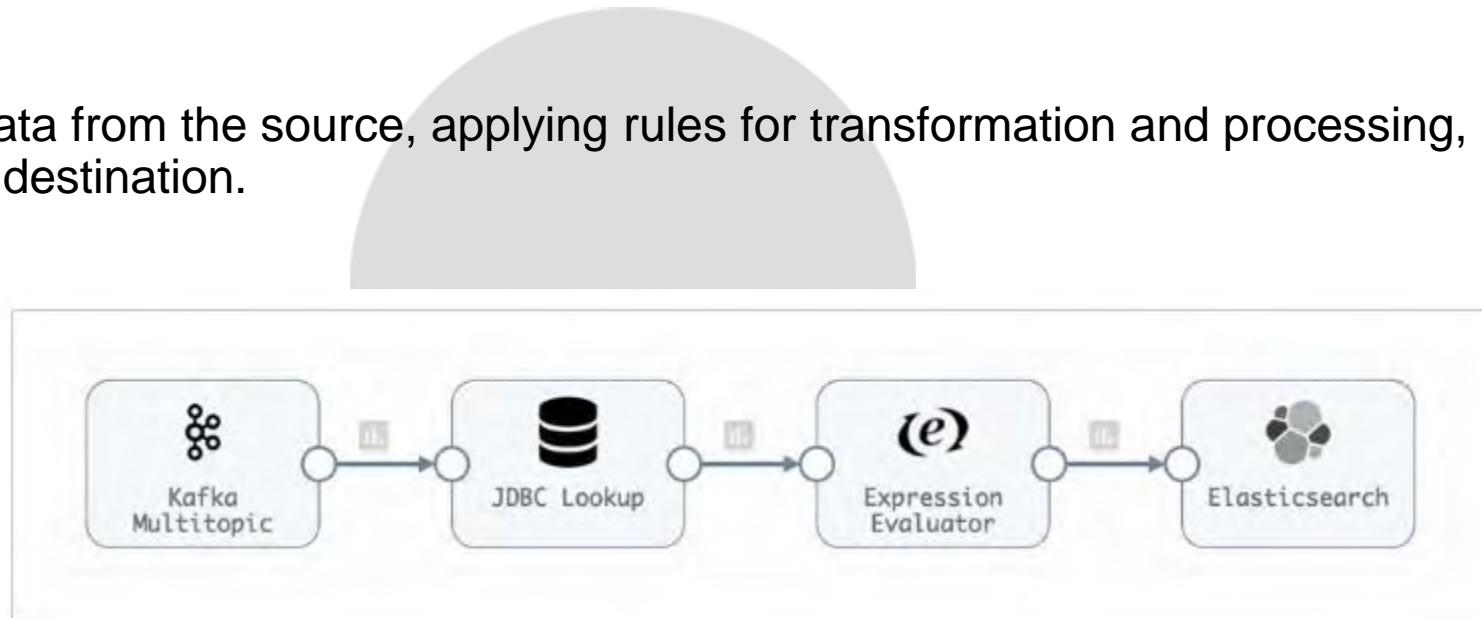
Data Pipelines - I

Pravin Y Pawar

Data Pipeline

What?

- Series of steps that allow data from one system to move to become useful in another system
 - particularly analytics, data science, or AI and machine learning systems
- Works by pulling data from the source, applying rules for transformation and processing, then pushing data to its destination.



Data Pipeline(2)

Purpose

- Lot of data out there!
- Data pipelines transform raw data into data ready for analytics, applications, machine learning and AI systems
 - keep data flowing to solve problems, inform decisions
- Used to:
 - Deliver sales data to sales and marketing for customer 360
 - Link a global network of scientists and doctors to speed drug discovery
 - Recommend financial services to help a small business owner thrive
 - Track COVID-19 cases and inform community health decisions
 - Combine diverse sensor data with AI for predictive maintenance
- With so much work to do, data pipelines can get pretty complicated pretty fast.

Data Pipeline(3)

Benefits

- Data pipelines done right have incredible advantages for companies and organizations, and the work they do.
- Self-service Data
 - Can be created ad hoc by data scientists and business analysts unstick the IT bottleneck
 - Means when people have brilliant ideas, they can test them, fail fast, and innovate faster
- Accelerate Cloud Migration and Adoption
 - Helps to expand cloud presence and migrate data to cloud platforms
 - Cloud computing helps to feed many new use cases at processing speeds, cost-effectiveness, and bursting capacity unheard of in traditional on-premises data centers
 - Teams can take advantage of rapid innovation happening on those cloud platforms such as
 - natural language processing
 - sentiment analysis
 - image processing
 - and more.
- Real-time Analytics and Applications
 - Real-time or near real-time functionality in consumer and business applications puts the pressure on data pipelines
 - to deliver the right data, to the right place, right now
 - Streaming data pipelines deliver continuous data to real-time analytics and applications

Data Pipeline(4)

Challenges

- Always Under Construction Data
 - Building and debugging data pipelines takes time
 - Have to align with the schema, set sources and destinations, check work, find errors, and back and forth until one can finally go live
 - by which time the business requirements may have changed again
 - That's why many data engineers have such a backlog of work
- Out of Order Data Pipelines
 - Even a small change to a row or a table can mean hours of rework, updating each stage in the pipeline, debugging, and then deploying the new data pipeline
 - Data pipelines often have to go offline to make updates or fixes
 - Unplanned changes can cause hidden breakages that take months of engineering time to uncover and fix
 - These unexpected, unplanned, and unrelenting changes are referred to as “data drift”.
- Build It and They Will Come
 - Data pipelines are built for specific frameworks, processors, and platforms.
 - Changing any one of those infrastructure technologies to take advantage of cost savings or other optimizations can mean weeks or months of rebuilding and testing pipelines before deployment.

Data Pipeline(5)

Working

- When a data pipeline is deployed and running
 - pulls data from the source
 - applies rules for transformation and processing
 - then pushes data to its destination
- Data sources
 - handle data in very different ways and might include applications, messaging systems, data streams, relational and NoSQL databases, cloud object storage, data warehouses, and data lakes
 - data structure varies significantly, depending on the source
- Transformations
 - Changes to data structure, format, or values as well as calculations and modifications to the data itself
 - A pipeline might have any number of transformations embedded to prepare data for use or route it correctly
- Destinations
 - systems where the data is ready to use, put directly into use, or stored for potential use
 - include applications, messaging systems, data streams, relational and NoSQL databases, data warehouses, data lakes, and cloud object storage

Data Pipeline(6)

Working

- 5 Common Data Pipeline Sources
 - JDBC
 - Oracle CDC
 - HTTP Client
 - HDFS
 - Apache Kafka
- Common Transformations
 - Masking PII (personally identifiable information) for protection and compliance
 - Converting data type for fields
 - Calculating based on a formula or expression
 - Renaming fields, columns, and features
 - Joining or merging datasets
 - Converting data formats (JSON to Parquet, for example)
 - Generating Avro Schema and other schema types on the fly
 - Handling Slowly Changing Dimensions
- 5 Common Data Pipeline Destinations
 - Apache Kafka
 - JDBC
 - Snowflake
 - Amazon S3
 - Databricks

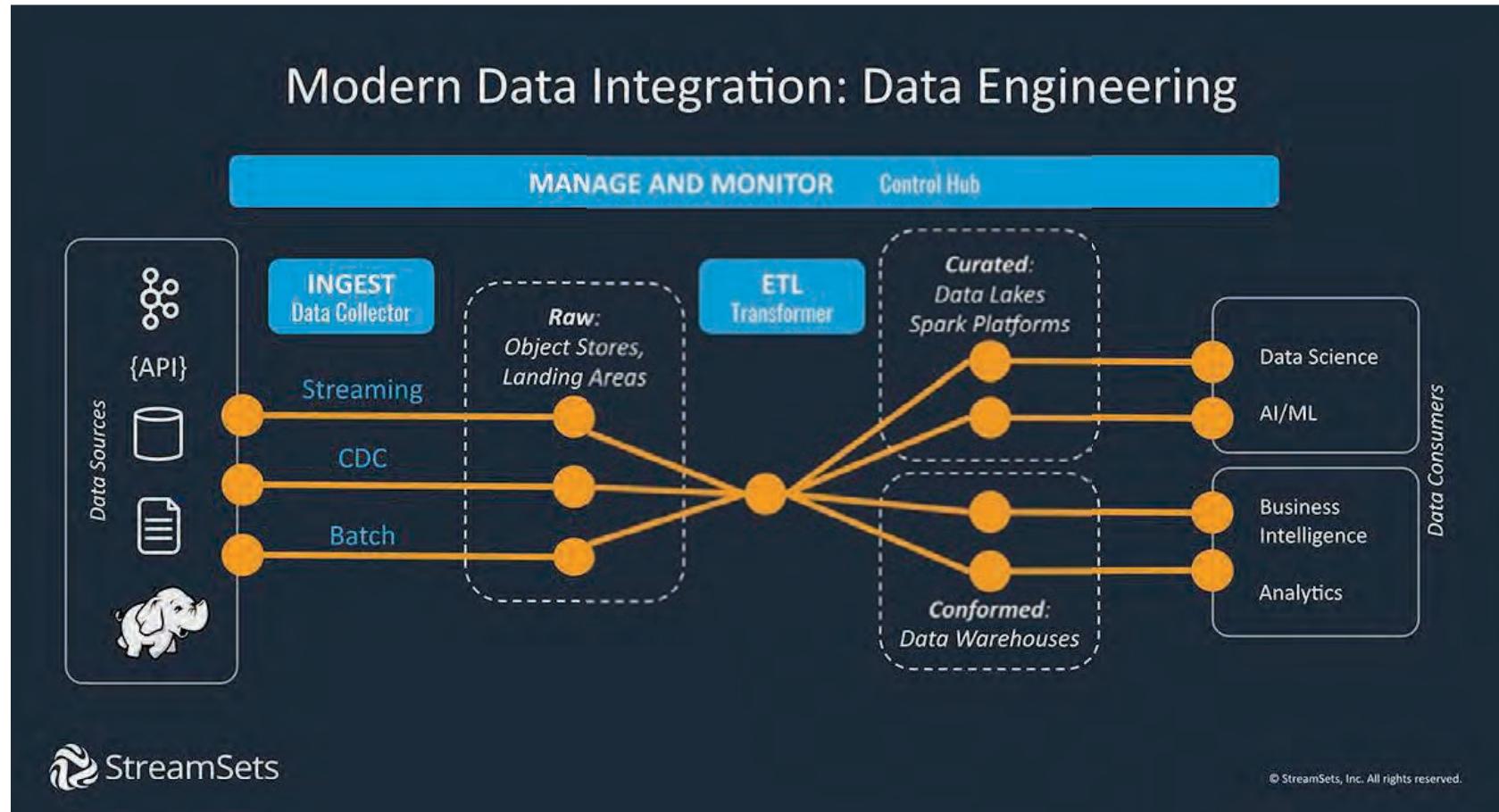
Data Pipeline(7)

Architectures

- Batch Data Pipeline
 - move large sets of data at a particular time or in response to a behavior or when a threshold is met
 - often used for bulk ingestion or ETL processing
 - might be used to deliver data weekly or daily from a CRM system to a data warehouse for use in a dashboard for reporting and business intelligence
- Streaming Data Pipeline
 - flow data continuously from source to destination as it is created
 - used to populate data lakes or as part of data warehouse integration, or to publish to a messaging system or data stream
 - used in event processing for real-time applications
 - streaming data pipelines might be used to provide real-time data to a fraud detection system and to monitor quality of service.
- Change Data Capture Pipeline (CDC)
 - used to refresh data and keep multiple systems in sync
 - Instead of copying the entire database, only changes to data since the last sync are shared
 - can be particularly useful during a cloud migration project when 2 systems are operating with the same data sets

Data Pipeline(8)

Architectures



Data Pipeline Tools

- Building a single pipeline for a single purpose at a given time is no problem
 - Can grab a simple tool for setting up a data pipeline or hand code the steps
- But how to scale that process to thousands of data pipelines in support of increasing demand for data across organization, over months or years?
 - When considering data pipeline tools, it is important to think ahead to where your data platform is headed.
- Questions to be pondered upon
 - Are you grabbing data from one place to put it somewhere else? Or do you need to transform it to fit the downstream analytics requirements?
 - Is your data environment stable and fully under your control? Or is it dynamic and pulling data from systems or apps outside of your control?
 - Will the pipeline move data once, for a short-term analysis project? Or will the pipelines you build need to be operationalized to handle data flows over time?

Data Pipeline Tools(2)

Data Ingestion and Data Loading Tools

- These tools make data pipelines easy to set up, easy to build, and easy to deploy solve the “under construction” issue
 - but only if you can count on your data scientists to do the data prep work later
- Only support the most basic transformations and work best for simply copying data
 - Instead of being intent-driven, these data pipelines are rigid and embedded with the specifics of data structure and semantics
 - Instead of adapting to change when data drift happens, they have to be rebuilt and deployed again

Data Pipeline Tools(3)

Data Integration, Data Transformation Platforms

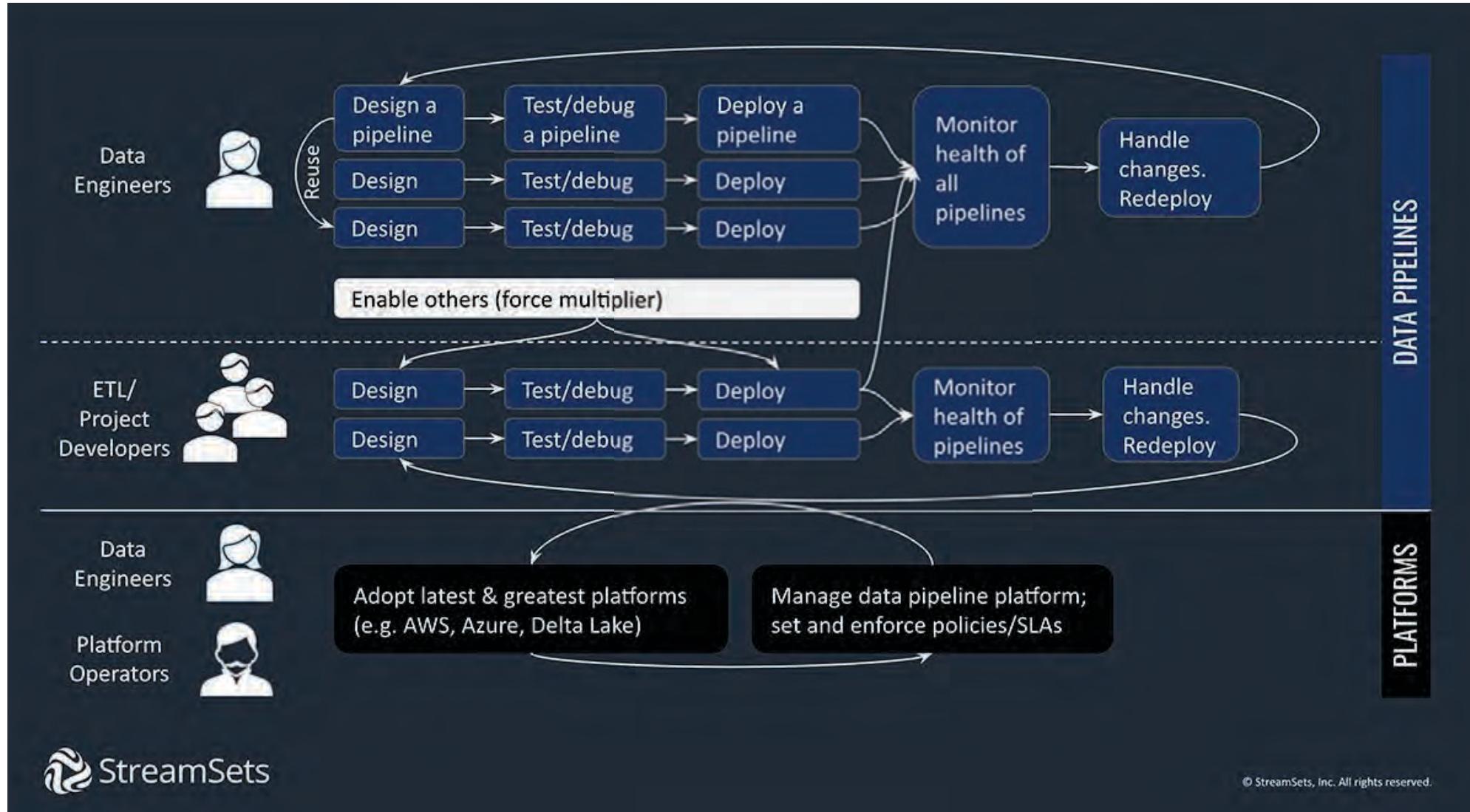
- More complex data integration or ETL software may have a solution for every possible scenario
 - with hundreds of connectors, integrations, and transformations
- But these platforms were designed for an era when data drift hardly happened
 - Things stayed the same for years
 - At the first hint of change, these data pipelines break and require massive rework
 - The demands for digital transformation are moving fast and planning for every possible outcome may not be possible

Data Pipeline Tools(4)

Data Engineering Platforms

- Third way!
- A data engineering platform builds smart data pipelines according to DataOps principles
 - Smart data pipelines abstract away the “how” so you can focus on the what, who, and where of the data
 - This is the fundamental difference between data integration and data engineering
- Instead of being perpetually under construction, out of order, or limited to a single platform
 - smart data pipelines allow you to move fast with confidence that your data will continue to flow with little to no intervention
- DataOps tools allow you to:
 - Design and deploy data pipelines in hours, not weeks or months
 - Build in as much resiliency as possible to handle changes
 - Adopt to new platforms by pointing to them, a task that takes minutes not months

Data Pipeline People



Reference

[Smart Data Pipelines: Tools, Techniques, and Key Concepts](#)



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Data Pipelines - II

Pravin Y Pawar

Agenda

- Data Analytics Pipelines
 - Need and Origin
 - Use cases
 - Types
 - Testing



Data Analytics Pipelines



Immature Data and Analytics Processes

Drawing Swords!

- Though data is a critical business asset
 - most organizations still don't have mature processes for converting data into insights that drive business value
- The development of data and analytics pipelines,
 - is still a handcrafted and largely non-repeatable process with minimal reuse,
 - managed by individuals working in isolation with different tools and approaches
- The result is
 - a plodding development environment that can't keep pace with the demands of a data-driven business
 - an error-prone operational environment that is slow to respond to change requests
- **Drawing Swords**
 - The lack of efficiency on the development side leads to a lack of effectiveness on the business side
 - Business and IT draw swords and go to war instead of partnering

Data Pipelines

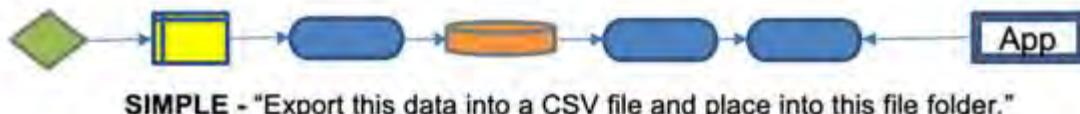
Origin

- Emerged from the big data community
- Data lakes created a feeding frenzy among developers, data scientists, and data analysts
 - gained unfettered access for the first time
 - to a nearly limitless supply of raw data sourced
 - from a multiplicity of internal and external systems
- Created a wide diversity of **workflows and programs**—or **data pipelines**—to process that data
 - (e.g., clean, filter, aggregate, move, load)
 - for a variety of purposes and use cases

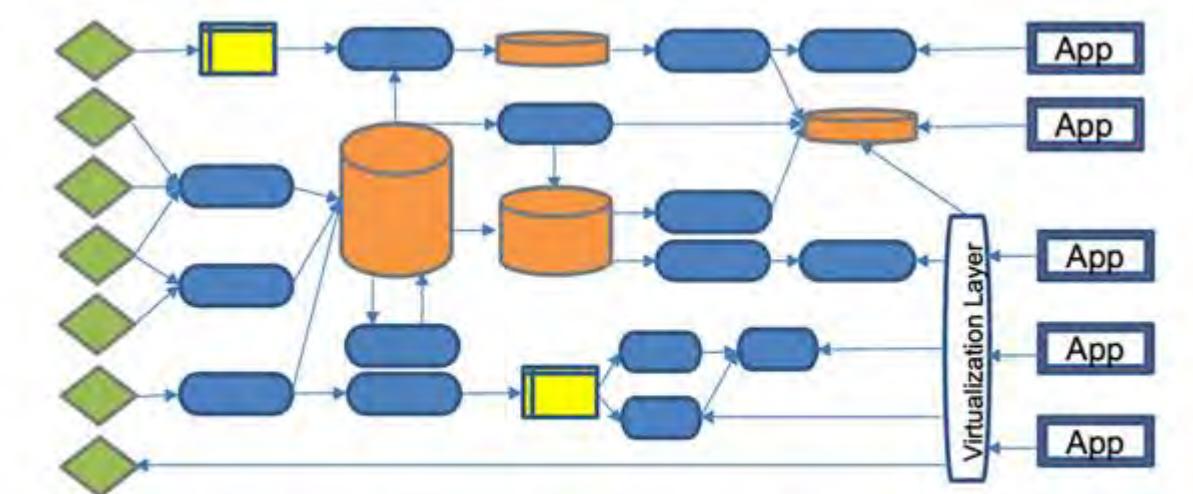
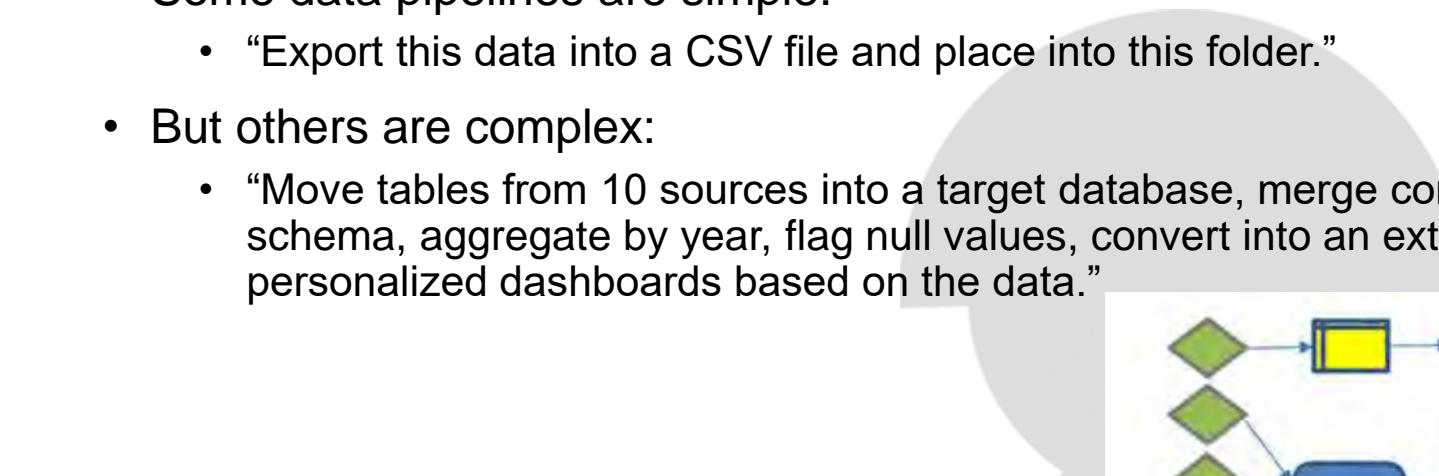
Data Pipelines(2)

Simple vs Complex

- Represents the encoded flow of data from source to consumption
- Some data pipelines are simple:
 - “Export this data into a CSV file and place into this folder.”
- But others are complex:
 - “Move tables from 10 sources into a target database, merge common fields, array into a dimensional schema, aggregate by year, flag null values, convert into an extract for a BI tool, and generate personalized dashboards based on the data.”



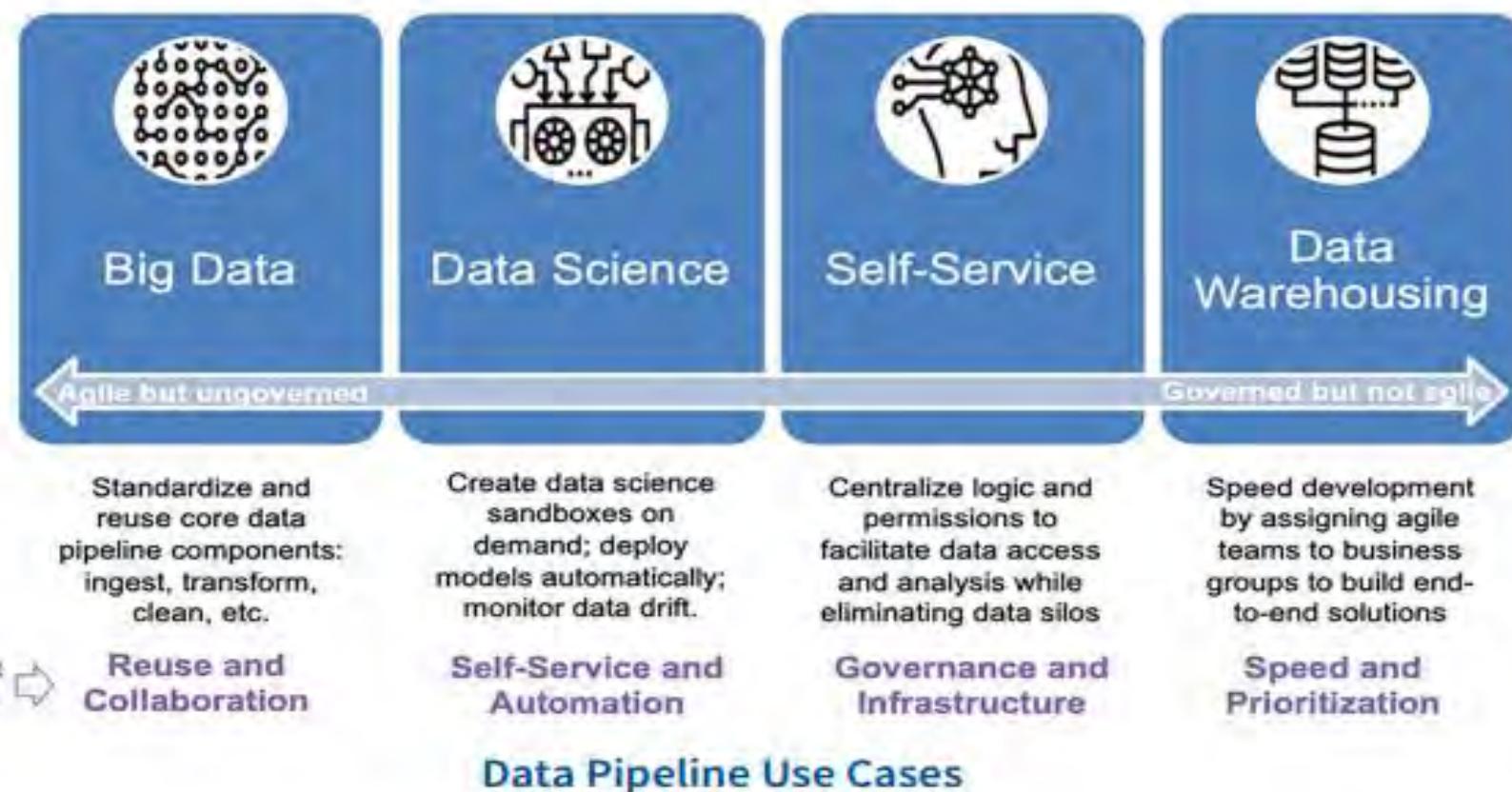
SIMPLE - "Export this data into a CSV file and place into this file folder."



COMPLEX - "Move tables from 10 sources into a target database, merge common fields, array into a dimensional schema, aggregate by year, flag null values, convert into an extract for a BI tool, and generate personalized dashboards based on the data."

Data Pipelines(3)

Use Cases



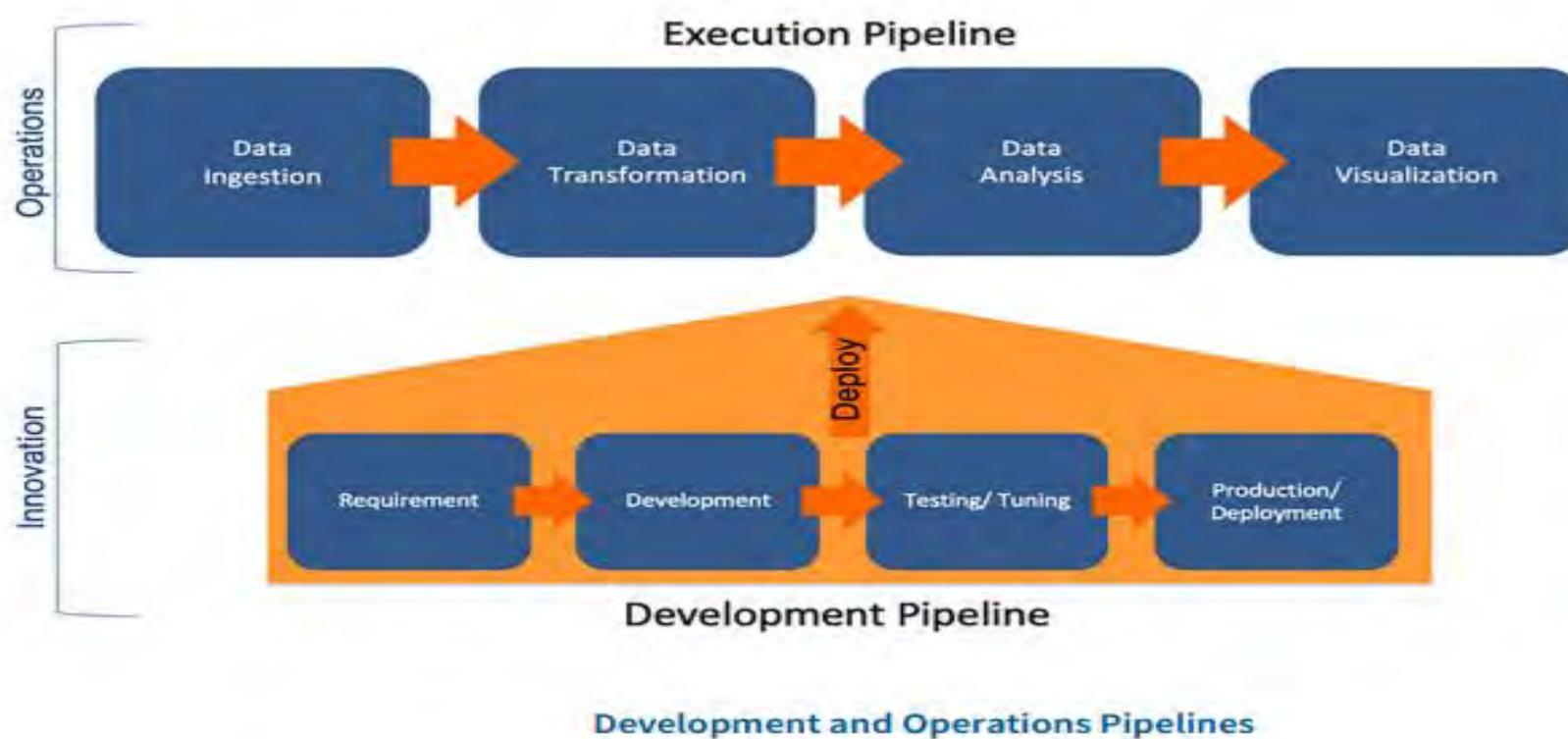
Data Pipelines(4)

Two types

- A development pipeline
 - creates code to process data for a new pipeline
- An execution pipeline
 - runs the pipeline in production
- In most IT shops
 - these are managed by different teams, creating tremendous inefficiencies
 - developers can build or change code without being responsible for its downstream impacts
 - inevitably increases error rates, delays delivery and frustrates business users

Data Pipelines(4)

Development and Execution Pipelines



Data Pipelines(5)

Generic Pipeline (Major)

- Although most data analytics pipelines are fairly complex, with dozens or hundreds of steps, most consist of the following stages:
- Data Ingestion
 - Data from various sources is extracted, validated, and loaded into downstream systems.
- Data Transformation
 - Data is cleansed, enriched, integrated, and modeled to support the target application or data structure.
- Data Analysis
 - Data is analyzed refined, and packaged in data models to provide certain insights or fulfill a task.
- Data Visualization
 - Data is visualized in a suitable manner, e.g., a static report or an interactive dashboard.

Data Pipelines(6)

People and Tools

- Data analytics pipelines
 - often span multiple functions supported by different specialists
 - who use a variety of tools and technologies
- A data engineer
 - uses ETL or data preparation tools
 - to extract data from a source and load it to a central data warehouse or data set
- A BI developer
 - uses a visualization tool
 - to build a chart, report or dashboard which is consumed by business users
- A data scientist
 - might then use Python or R
 - to build a predictive model that provides data for a business application managed by a software developer

Data Pipelines(7)

Micro Pipeline

- To manage this complexity, it helps to divide individual stages into micro-pipelines
 - to ensure that changes can be made to each step without disrupting the entire flow
- The micro-pipelines consist of the following steps:
- Requirement
 - Agile methods, such as Scrum and Kanban, elicit requirements from stakeholders in an iterative manner
 - A requirement could be the integration of a new data source in an ETL process or the addition of a new field to a database table
- Development
 - In data and analytics, development does not necessarily mean programming
 - it can comprise tasks like changing a data model or reconfiguring a dash-board
 - Requires a high level of automation
 - e.g., automatically generating database queries when a model is changed, to avoid repetitive manual work

Data Pipelines(8)

Micro Pipeline

- Testing/Tuning
 - Testing is often neglected in data and analytics, as building an adequate testing environment in complex analytics solutions can be difficult
 - Many tools simply lack testing functions
- Production/Monitoring
 - After testing, changes are deployed to production
 - Should be a fully automated, documented, and easily reversible
- Orchestration and Automation
 - A key function is to orchestrate and automate the flow of data and code between people and tools in an efficient manner
 - that ensures clean handoffs and minimal errors and disruptions
 - With complex pipelines, this can be challenging, making orchestration and automation key facilities in any implementation

Testing

Need

- Tests are central to both types of pipelines
 - In development pipelines, the data is fixed but code is variable
 - In production the data is variable and code is fixed
- **Development tests the code, and production tests the data!**
- Teams must build tests for both types of pipelines
 - run them continuously in both development and production
 - to ensure that changes don't adversely affect outcomes
 - which accelerates development and minimize operational delays
- Unfortunately, too many development shops make changes, perform minimal tests,
 - then cross their fingers that the changes don't break something!

Testing(2)

Types of Tests

- Finding issues before internal customers do is critically important for a development team
- Three types of tests
 - Data input tests prevent bad data from entering a new node in a pipeline stage
 - Business logic tests validate that data matches business assumptions
 - Data output tests verify that a pipeline stage executed properly

Types of Tests	
Inputs	Verifying the inputs to an analytics processing stage Count Verification - Check that row counts are in the right range, ... Conformity - US Zip5 codes are five digits, US phone numbers are 10 digits, ... History - The number of prospects always increases, ... Balance - Week over week, sales should not vary by more than 10%, ... Temporal Consistency - Transaction dates are in the past, end dates are later than start dates, ... Application Consistency - Body temperature is within a range around 98.6F/37C, ... Field Validation - All required fields are present, correctly entered, ...
Business Logic	Checking that the data matches business assumptions Customer Validation - Each customer should exist in a dimension table Data Validation - 90 percent of data should match entries in a dimension table
Output	Checking the result of an operation, for example, a cross-product join Completeness - Number of customer prospects should increase with time Range Verification - Number of physicians in the US is less than 1.5 million

From "Build Trust through Test Automation and Monitoring," DataKitchen, September 6, 2018.

References

- The Ultimate Guide to DataOps Product Evaluation and Selection Criteria
- DataOps: Industrializing Data and Analytics Strategies for Streamlining the Delivery of Insights
- IBM DataOps How and Why Whitepaper



Thank You!



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

DataOps

Pravin Y Pawar

Agenda

- DataOps
 - Approach
 - Foundations and Requirements
 - Technology Framework
 - Tools and vendor landscape
 - Benefits and Pitfalls





DataOps

DataOps

New Approach

- Data Analytics pipelines becoming more complex and development teams grow in size
 - organizations need better collaboration and development processes
 - to govern the flow of data and code from one step of the data lifecycle to the next
 - – from data ingestion and transformation to analysis and reporting
- DataOps
 - is an emerging set of practices, processes, and technologies
 - for building and enhancing data and analytics pipelines
 - to meet business needs quickly
- The goal is to increase agility and cycle times,
- While
 - reducing data defects,
 - giving developers and business users greater confidence in data analytics output

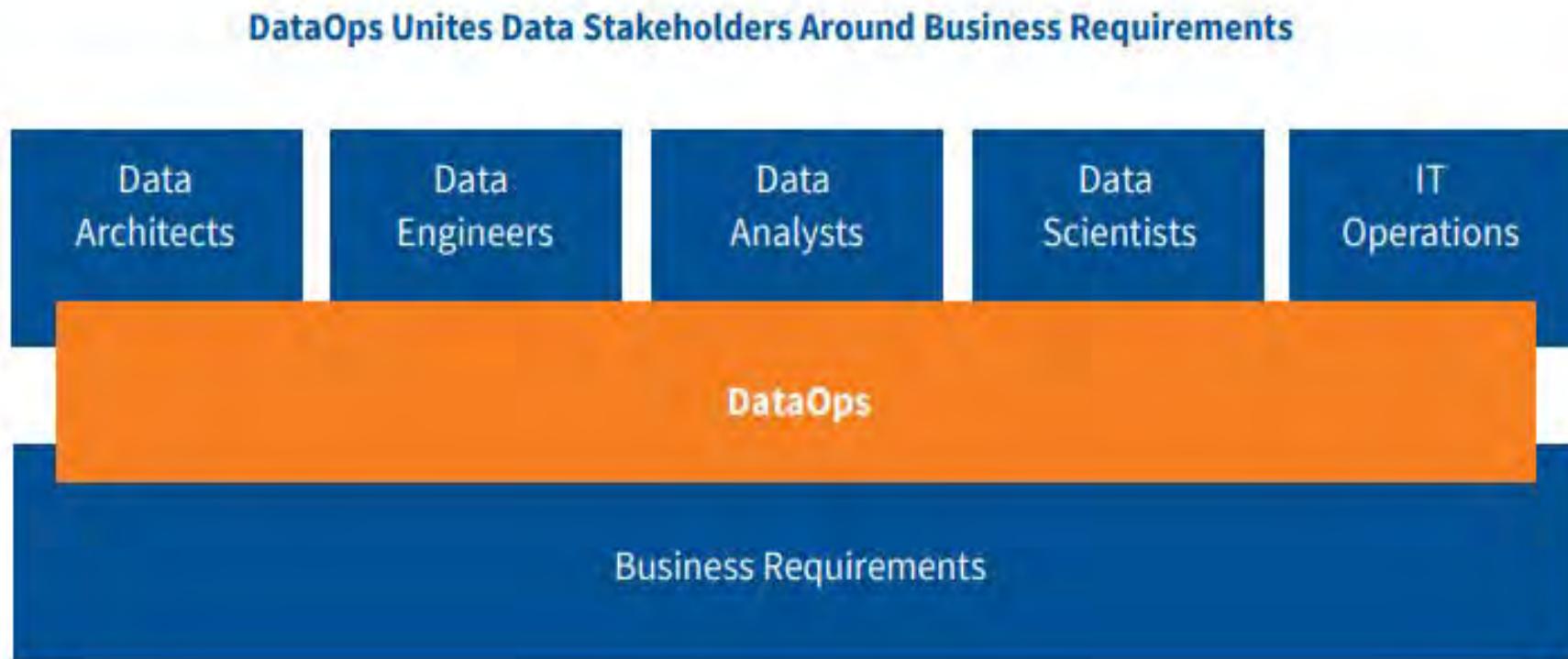
DataOps(2)

Foundations

- Builds on concepts popular in the software engineering field,
 - such as agile, lean, and continuous integration/continuous delivery
- but addresses the **unique needs of data and analytics environments**,
 - including the use of multiple data sources and varied use cases that range from data warehousing to data science
- Relies heavily on
 - test automation
 - code repositories
 - collaboration tools,
 - orchestration frameworks
 - and workflow automation
- to accelerate delivery times while minimizing defects

DataOps(3)

Data Stakeholders



DataOps(4)

DataOps for Data Warehousing and Data Management

Table 1. DataOps for Data Warehousing and Data Management

Current Challenges:	DataOps' Answers:
<ul style="list-style-type: none">• Complex data landscapes and processes with heterogeneous data sources and tools that are hard to manage.• A lack of agility makes it hard to respond to rapidly changing business requirements.	<ul style="list-style-type: none">• Analytics pipelines that streamline processes, improve collaboration, and establish a culture of continuous improvement.• Use of automation to increase agility and speed and free resources for more valuable activities.

DataOps(5)

DataOps for Dashboards and Reports

Table 2. DataOps for Dashboards and Reports

Current Challenges:	DataOps' Answers:
<ul style="list-style-type: none">• Manage a growing number of reports and dashboards over the entire lifecycle.• Understand what the business needs and quickly respond to changing requirements.	<ul style="list-style-type: none">• Establish an end-to-end responsibility to bridge the gap between data and business.• Ensure relevancy with short cycles, fast feedback, and data-driven improvements.• Improve speed, quality, and scalability with defined analytics pipelines.

DataOps(6)

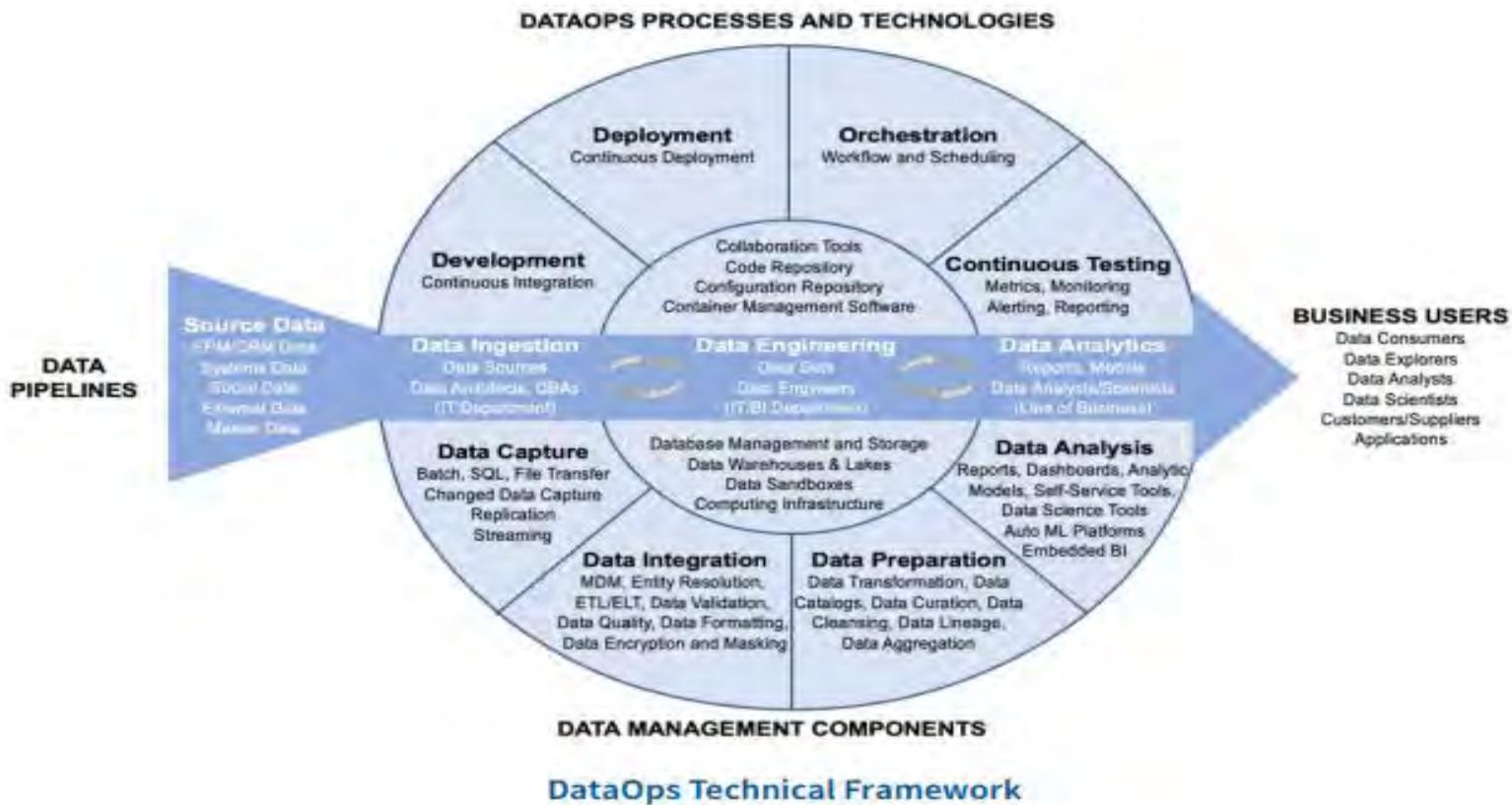
DataOps for Data Science

Table 3. DataOps for Data Science

Current Challenges:	DataOps' Answers:
<ul style="list-style-type: none">• Transform statistical models from an experimental stage to production so they deliver ROI.• Alignment among data scientists, product owners, and data engineers.	<ul style="list-style-type: none">• Establish analytics pipelines that keep the data science process flexible.• Use tools to orchestrate stakeholders and technologies throughout the entire data science process.• Combine different skillsets with cross functional thinking.

DataOps(7)

Technology



DataOps(8)

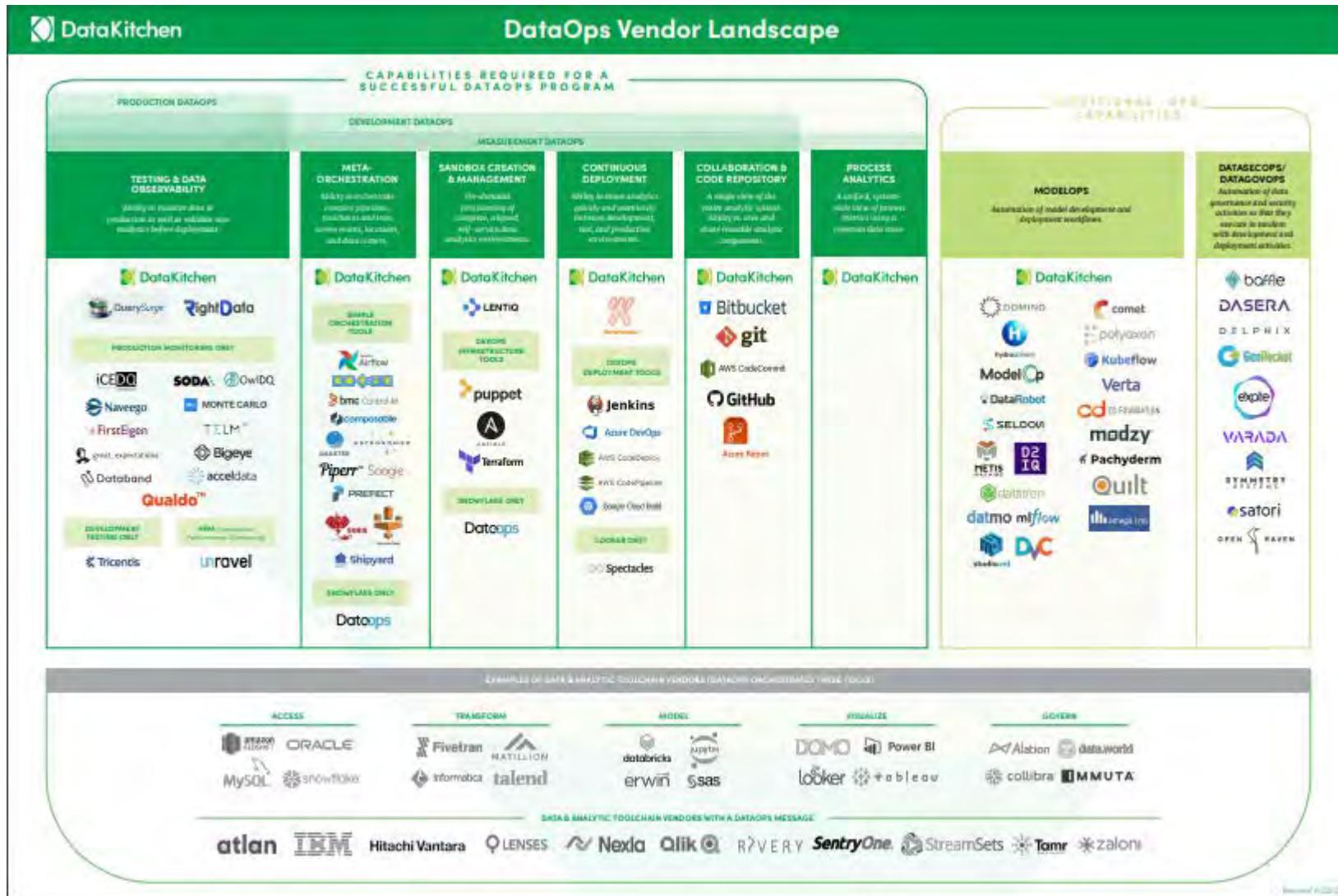
Tools

- There are five categories of DataOps products on the market today:
 - All-in-one tools
 - Orchestration tools
 - Component tools
 - Case-specific tools
 - Open source tools



DataOps(9)

Vendor Landscape



DataOps(10)

Benefits of DataOps

- Improve Collaboration and Communication
 - comes with a change in culture that promotes collaboration, trust, and responsibility
 - goals are to blur the lines between departments and functions,
 - encourage the exchange of knowledge, reduce conflicts, and eventually increase productivity
- Accelerate Time to Production
 - A major driver for DataOps is speed
 - Idea of streamlined and largely automated analytics pipelines helps deliver new features and insights quickly and reduces manual effort
 - Short feedback and testing cycles help speed up reactions to changing business requirements and increase flexibility
- Increase Quality and Reliability
 - Well-defined analytics pipelines enhance both speed and robustness
 - For instance, multiple stages of automated and manual tests prevent the deployment of flawed updates
 - Besides, DataOps also includes monitoring of rolled-out changes to identify bottlenecks and potential issues
 - Lastly, the convergence of different roles helps align changes throughout various stages,
 - such as when a data engineer is informed about the later cleansing issues encountered by a data scientist

DataOps(11)

Common Pitfalls of DataOps

- The following common pitfalls should be considered on the way to a DataOps culture
- Acceptance:
 - **Don't make it technical!**
 - DataOps has the same burden as other related ideas in software engineering, which are often characterized as technical Knick knacks
 - It is important to show that DataOps is not a technology trend, but a business trend
 - Communicate the holistic idea of DataOps, get everybody on board early, and show that business value is the goal, rather than having a fancy tech stack
- Engineering:
 - **Keep it simple!**
 - DataOps combines many concepts which can be extended almost endlessly, running the risk of over-complication
 - Always take care to balance effort and benefit
 - This applies to both technical solutions, such as unnecessary tests and overly complex deployment processes, and to organizational structures, which can easily become burdened with too many roles, feedback loops, and overly rigid processes.

References

- The Ultimate Guide to DataOps Product Evaluation and Selection Criteria
- DataOps: Industrializing Data and Analytics Strategies for Streamlining the Delivery of Insights
- IBM DataOps How and Why Whitepaper



Thank You!



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Modern Data Stack

Pravin Y Pawar

A typical TDS setup

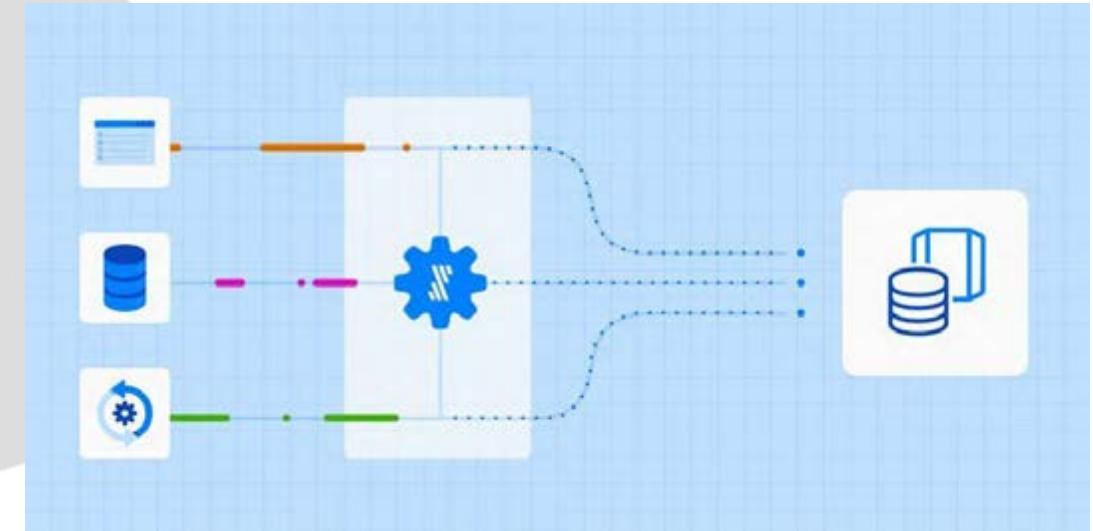
Three major problems

- Long turn-around time to untangle and set up infrastructure
 - Companies making use of **on-premise infrastructure** are responsible for all the costs associated with it
 - such as the army of engineers required to keep everything maintained and running smoothly.
 - As setup is so deeply interconnected, what may seem like **a minor change might break other parts of the system**
 - Finding the exact logical coupling between the systems requires **a lot of work-hours to analyze before any improvements** can be made to the existing landscape
- Slow response to new information
 - As the company grows, so does its **data and computational power needs**
 - **Very costly** in terms of resources and time when it comes to **scaling out (expanding) on-premises infrastructure**
 - leads to a limit in how much computational power there is to analyze data
 - data pipelines can take hours to complete, a problem that is compounded as the organization grows
 - **A TDS requires slow ETL (Extract, Transform, Load) operations before newly ingested data can conform to the rest of the data model**
 - A new data update can take weeks and many hours of refactoring before insights appear
 - By the time the data is ready, the organization is unable to act in time resulting in missed opportunities.
- Expensive journey to insights
 - A lot of the **report generation is manually done**, especially when the data is coming from different sources
 - The report is manually generated, manually cleaned, and manually transferred to Excel
 - Leads to **errors being made**, time being taken away from other business-critical tasks, and the inability to scale.
 - Analysts are unable to efficiently perform their roles due to the complex landscape

[Source: neptune](#)

Modern Data Stack

- A radically new approach to data integration saves engineering time, allowing engineers and analysts to pursue higher-value activities!
- The modern data stack (MDS) is a **suite of tools used for data integration**. These tools include, in order of how the data flows:
 - a fully managed ELT data pipeline
 - a cloud-based columnar warehouse or data lake as a destination
 - a data transformation tool
 - a business intelligence or data visualization platform.
- The goal of an MDS is to analyze business's data to proactively uncover new areas of opportunity and improve efficiency



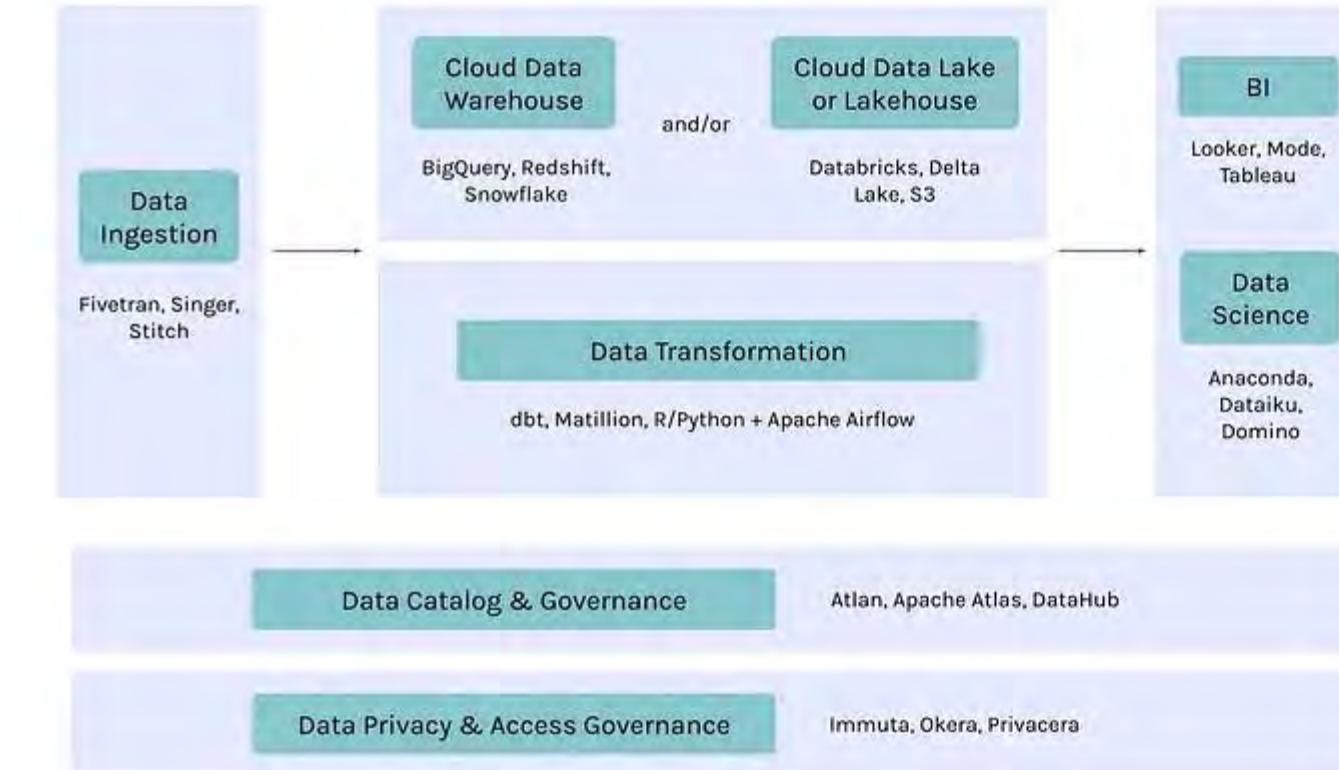
[Source: fivetran](#)

The 3 characteristics of a modern data platform

- A modern platform should be...
 - Easy to set up
 - no lengthy sales process, demo calls, and implementation cycles. Just login, pay via credit card, and go!
 - Pay as you go
 - no up-front payments and million dollar licensing fees. The “modern” stack is all about putting power in the hands of the consumer, i.e. paying only for what you use
 - Plug and play
 - the modern data stack will continue to evolve and innovate, and the best of breed tools do not enforce “lock in” like tools in the legacy era, and are instead built on open standards & APIs allowing easy integration with the rest of the stack

[Source: the-building-blocks-of-a-modern-data-platfor](#)

The key building blocks of a modern data platform



Source: atlan

The key building blocks of a MDS(2)

Modern data ingestion

- Data ingestion is likely where efforts to build out a modern data platform get started
 - i.e. how to bring data from a variety of different data sources and ingest it into core data storage layer?
- The majority of data pipelines capture data in batch, using SQL queries or extract routines, but this is changing
 - Many companies now prefer a streaming-first architecture that captures and processes data in real time in a continuous stream
 - Streaming tools use change data capture, replication, and/or publish/subscribe messaging systems to capture events as they happen, move them into the data environment, and analyze them.
- Some key modern data ingestion tools:
 - SAAS tools: Fivetran, Hevo Data, Stitch
 - Open-source tools: Singer, StreamSets
 - Custom data ingestion pipelines built on orchestration engines like Airflow

The key building blocks of a MDS(3)

Modern data storage and processing

- The data storage and processing layer is fundamental to the modern data platform
- Architecture is evolving, typically see **3 kinds of tools or frameworks**:
- Data warehouses
 - The cornerstone of this architecture is a modern data warehouse
 - generally the system of choice for analysts since they optimize compute and processing speed
 - Key data warehouse tools include **BigQuery, Redshift, and Snowflake**.
- Data lakes
 - Refers to data being stored on an **object storage like Amazon S3**, coupled with tools to process the data such as **Spark**
 - cheap storage systems are often used to store vast amounts of **raw or even unstructured data**
 - Key tools for data lakes:
 - Data storage: Amazon S3, Azure Data Lake Storage Gen2, Google Cloud Storage
 - Data processing engines: Databricks or Spark; Athena, Presto, or Starburst; Dremio
- Data lakehouses
 - One of the trends is seeing this year is the **long-awaited convergence of data warehouses and lakes**
 - help unify the siloed systems that most companies have created over the last decade
 - **a system design that combines the data management features such as ACID transactions and change data capture from a data warehouse with the low-cost storage of a data lake.**

The key building blocks of a MDS(4)

Modern data transformation

- Two core implementations seeing today for the data transformation layer
- For companies with data warehouse–first architectures
 - tools like dbt that leverage native SQL for transformation have emerged as the top choice for data transformation
- The other common implementation is using Airflow as an orchestration engine coupled with custom transformation in a programming language like Python
- Key tools for data transformation:
 - With data warehouses: dbt, Matillion
 - With an orchestration engine: Apache Airflow + Python, R, or SQL

The key building blocks of a MDS(5)

Modern business intelligence and analytics

- BI dashboards have been around for ages
 - but the latest breed of BI and analytics tools are built to fit within a larger modern data platform
 - generally more **self-service platforms** that allow users **to explore data**, rather than just consuming graphs and charts
- Modern BI tools include **Looker, Mode, Redash, Sigma, Sisense, Superset, and Tableau.**

The key building blocks of a MDS(6)

- While the modern data platform is great in some areas (super fast, easy to scale up, little overhead),
 - it struggles with bringing discovery, trust and context to data.
- Key tools for modern data cataloguing and governance:
 - SAAS tools: Atlan
 - Open-source tools: Apache Atlas, LinkedIn's DataHub, Lyft's Amundsen
 - In-house tools: Airbnb's Dataportal, Facebook's Nemo, Uber's Databook

The key building blocks of a MDS(7)

Modern data privacy and access governance

- A major challenge is to be able to manage privacy controls and access governance across entire stack
- While it's still **early**, there have been some **recent players and developments** in the space and players emerging with tools
 - that can act as an entitlement engine to **apply privacy and security policies across the data stack**
- Key tools for data privacy and access governance:
 - SaaS services: Immuta, Okera, Privacera
 - Open-source engines: Apache Ranger

Other modern data tools you should know about

Other modern data tools you should know about

- Each company uses data differently, so many have **added additional layers and tools for specific use cases**
- Real-time data processing tools
 - Companies that need real-time data processing generally add two additional types of tools to their data platform:
 - Real-time streaming pipelines: Confluent, Kafka
 - Real-time analytics: Druid, Imply
- Data science tools
 - Companies that have moved past BI and analytics onto strong predictive and data science analytics often add a specific data science tool (or tools) to their data stack:
 - Preferred by data scientists: Jupyter Notebooks
 - Other options: Dataiku, DataRobot, Domino, SageMaker
- Event collectors
 - Companies with a significant digital presence often add event collectors to log and store external events
 - Tools like Segment and Snowplow are key additions to their data ingestion stack.

Other modern data tools you should know about(2)

- Data quality tools
 - This space is still relatively nascent, but it's seeing a lot of activity nowadays.
 - seeing a few aspects of data quality incorporated throughout the data stack — data quality checks during profiling, business-driven quality rules, and unit testing frameworks within the pipeline
- Data profiling
 - getting engulfed by data catalog and governance tools like Atlan or open source profiling frameworks like Amazon Deequ
- Unit testing
 - Frameworks like the open-source Great Expectations are emerging and evolving, allowing unit tests to be written as a part of data pipelines itself.

Reference:

The Building Blocks of a Modern Data Platform

<https://towardsdatascience.com/the-building-blocks-of-a-modern-data-platform-92e46061165>



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

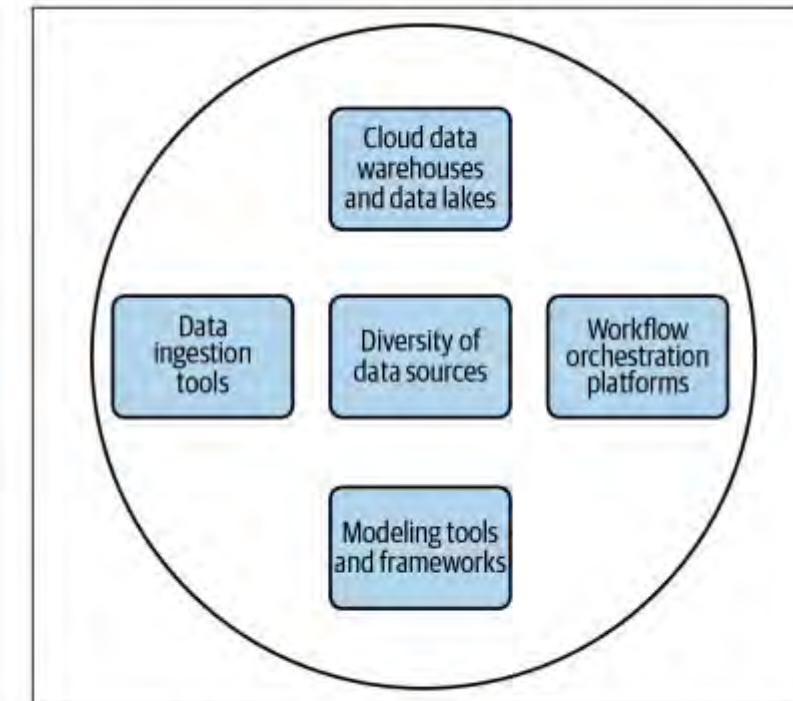
Modern Data Infrastructure

Pravin Y Pawar

Adapted from
Data Pipelines Pocket Reference
By James Densmore

Key components of modern data infrastructure

- No single right way to design analytics ecosystem or choose products and vendors
- Before deciding on products and design for building data pipelines,
 - it's worth understanding what makes up a modern data infrastructure

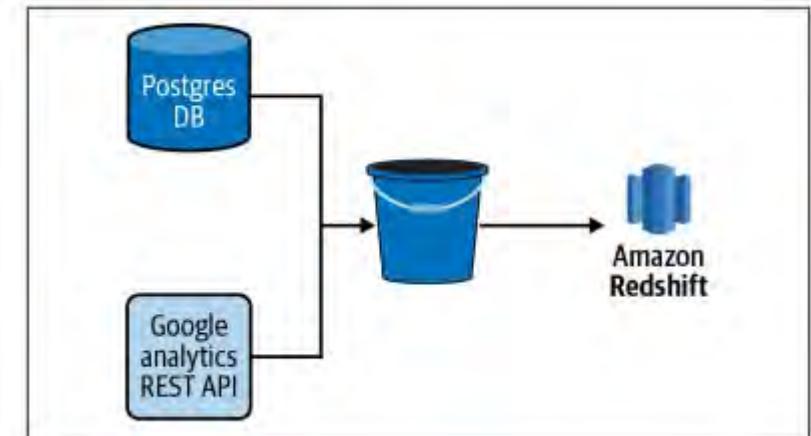
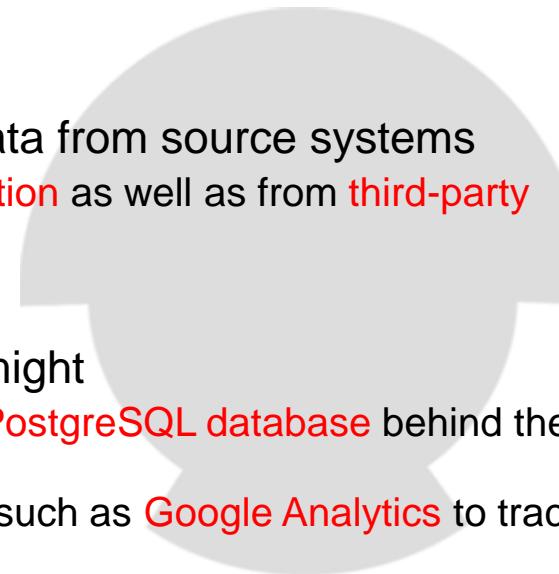


The key components of a modern data infrastructure.

Diversity of Data Sources

Source Systems

- The majority of organizations have **dozens**, if not hundreds, of **data sources**
 - that feed their analytics endeavors
- Typical for an analytics team to ingest data from source systems
 - that are **built and owned by the organization** as well as from **third-party tools and vendors**
- For example, an **ecommerce company** might
 - store data from their shopping cart in a **PostgreSQL database** behind their web app
 - also use a third-party web analytics tool such as **Google Analytics** to track usage on their website
 - **combination** of the two data sources is required to get a **full understanding** of customer behavior leading up to a purchase
- A data pipeline that ends with an analysis of such behavior starts with the ingestion of data from both sources.



Ingestion Interface and Data Structure

Data Ingestion - extracting data from one source and loading it into another

- When building a new data ingestion
 - Essential to know how to get data in and in what form
- First, what is the **interface to the data?**
 - A database behind an application, such as a Postgres or MySQL database
 - A layer of abstraction on top of a system such as a REST API
 - A stream processing platform such as Apache Kafka
 - A shared network file system or cloud storage bucket containing logs, comma-separated value (CSV) files, and other flat files
 - A data warehouse or data lake
 - Data in HDFS or HBase database
- In addition to the interface, **the structure of the data** will vary.
 - JSON from a REST API
 - Well-structured data from a MySQL database
 - JSON within columns of a MySQL database table
 - Semi-structured log data
 - CSV, fixed-width format (FWF), and other flat file formats
 - JSON in flat files
 - Stream output from Kafka
- Each interface and data structure presents its own challenges and opportunities

Cloud Data Warehouses and Data Lakes

- Three things transformed the landscape of analytics and data warehousing over the last 10 years
 - all related to the **emergence of the major public cloud providers** (Amazon, Google, and Microsoft):
 - The **ease of building and deploying** data pipelines, data lakes, warehouses, and analytics processing in the cloud
 - Continued **drop-in storage costs** in the cloud
 - The **emergence of highly scalable, columnar databases**, such as Amazon Redshift, Snowflake, and Google Big Query
- These changes breathed new life into data warehouses and introduced the concept of a data lake!
- A data warehouse is a database where data from different systems is stored and modeled to support analysis and other activities related to answering questions with it
 - Data in a data warehouse is structured and optimized for reporting and analysis queries
- A data lake is where data is stored, but without the structure or query optimization of a data warehouse
 - likely contain a high volume of data as well as a variety of data types
- There is a place for both data warehouses and data lakes in the same data ecosystem, and data pipelines often move data between both.

Data Ingestion Tools

- The need to **ingest data from one system to another** is common to nearly all data pipelines
 - data teams must contend with a diversity of data sources from which to ingest data from
- A number of commercial and open source tools are available in a modern data infrastructure.
 - Singer
 - Stitch
 - Fivetran
- Despite the prevalence of these tools,
 - some teams decide to **build custom code** to ingest data
 - Some even develop their **own frameworks**
- Reasons vary by organization but are often related to
 - **cost, a culture of building over buying, and concerns about the legal and security risks of trusting an external vendor.**
 - need to assess the value of a commercial solution is to **make it easier for data engineers to build data ingestions** into their pipelines or to **enable nondata engineers (such as data analysts) to build ingestions themselves**
- Data ingestion is traditionally both the extract and load steps of an ETL or ELT process
 - Some tools focus on just these steps, while others provide the user with **some transform capabilities** as well

Data Transformation and Modeling Tools

- Pipelines are also made up of tasks that transform and model data for new purposes
 - such as machine learning, analysis, and reporting
 - terms often used interchangeably
- Data transformation
 - Transforming data is a broad term that is signified by the T in an ETL or ELT process
 - A transformation can be something
 - as simple as converting a timestamp stored in a table from one time zone to another
 - a more complex operation that creates a new metric from multiple source columns that are aggregated and filtered through some business logic
- Data modeling
 - Data modeling is a more specific type of data transformation
 - A data model structures and defines data in a format that is understood and optimized for data analysis
 - A data model is usually represented as one or more tables in a data warehouse

Data Transformation and Modeling Tools(2)

- Data Transformation Tools
 - Some **data ingestion tools** provide some level of data transformation capabilities, but these are often quite **simple**
 - for the sake of protecting personally identifiable information (PII) it may be desirable to turn an email address into a hashed value
 - For more **complex data transformations and data modeling**, find it desirable to seek out tools and frameworks **specifically designed** for the task, such as dbt
 - Additionally data transformation is **often context-specific** and can be written in a **language** familiar to data engineers and data analysts, such as SQL or Python.
- Data modeling Tools
 - typically defined and written in **SQL** or via **point-and-click user interfaces**
 - SQL is a highly accessible language that is **common to both data engineers and analysts**
 - empowers the analyst to work directly with the data and optimize the design of models for their needs
 - used in nearly every organization, thus providing a familiar entry point for new hires to a team
- In most cases, choosing a transformation framework that supports building data models in SQL rather than via a point-and click user interface is desirable!

Workflow Orchestration Platforms

- As the complexity and number of data pipelines in an organization grows
 - it's important to introduce a workflow orchestration platform to data infrastructure
 - manages the scheduling and flow of tasks in a pipeline
- Imagine a pipeline with a dozen tasks ranging from
 - data ingestions written in Python
 - data transformations written in SQL
 - that must run in a particular sequence throughout the day
 - not a simple challenge to schedule and manage dependencies between each task
- There are numerous workflow orchestration platforms available to alleviate the pain
 - Apache Airflow, Luigi, and AWS Glue, are designed for more general use cases and are thus used for a wide variety of data pipelines
 - Kubeflow Pipelines, are designed for more specific use cases and platforms (machine learning workflows built on Docker containers in the case of Kubeflow Pipelines)

Customizing Data Infrastructure

- It's rare to find two organizations with exactly the same data infrastructure
 - Most pick and choose tools and vendors that meet their specific needs and build the rest on their own
 - many more come to market each year
- Depending on the culture and resources in organization,
 - may be encouraged to **build most of data infrastructure on their own**
 - or to rely **on SaaS vendors** instead
- Regardless of which way to lean on the **build versus-buy scale**
 - need to build the **high-quality data infrastructure** necessary to build **high-quality data pipelines**
- Most important is understanding constraints (dollars, engineering resources, security, and legal risk tolerance) and the resulting trade-offs!



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

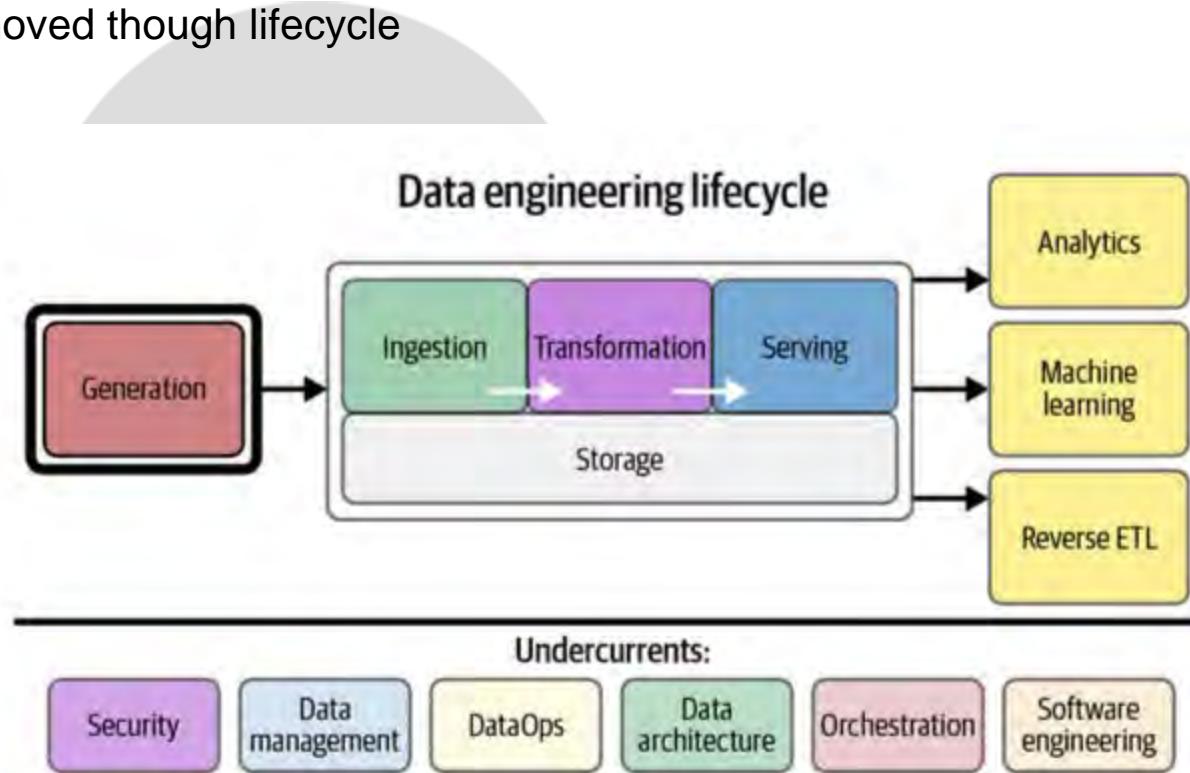
Data Storage Infrastructure

Pravin Y Pawar

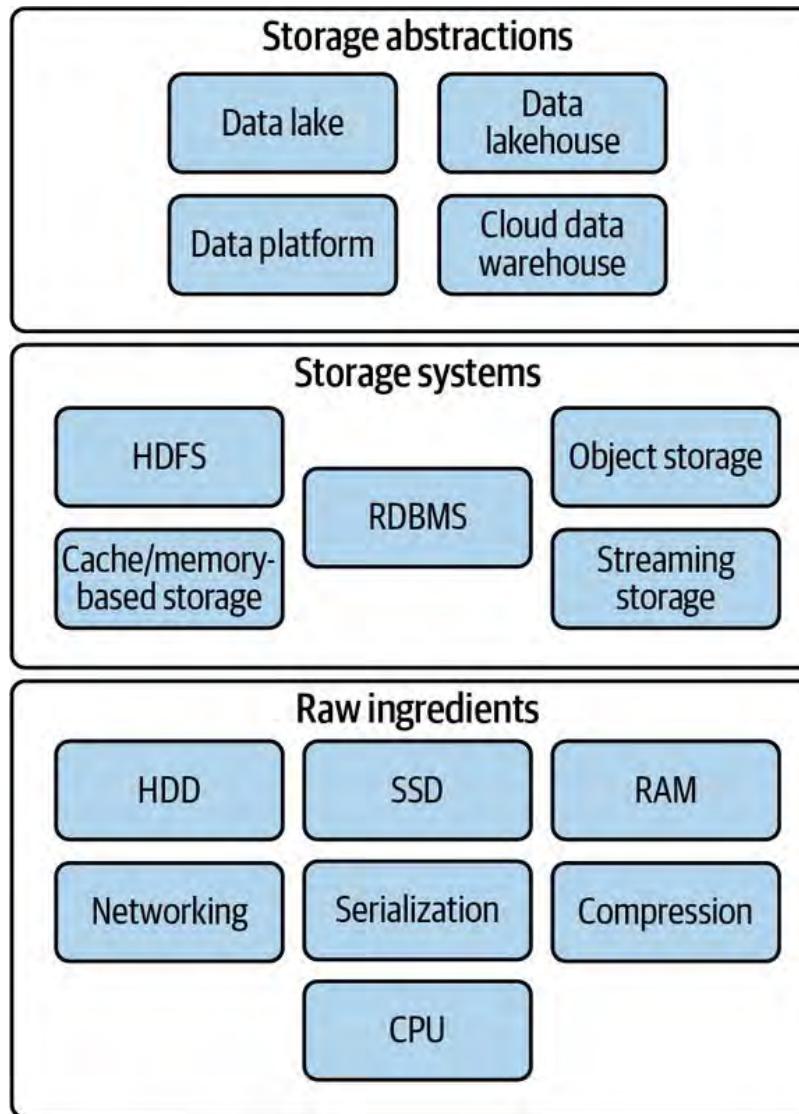
Adapted from
Fundamentals of Data Engineering
By Joe Reis and Matt Housley

Storage

- Cornerstone in the data engineering life cycle
 - Underlies major stages – ingestion, transformation and serving
 - Get stored many times when it moved though lifecycle



Storage Layering



Raw Ingredients of Data Storage

- Hardware
 - Magnetic Disk Drive
 - Solid-State Drive
 - Random Access Memory
 - Networking and CPU
- Associated features
 - Serialization
 - Compression
 - Caching

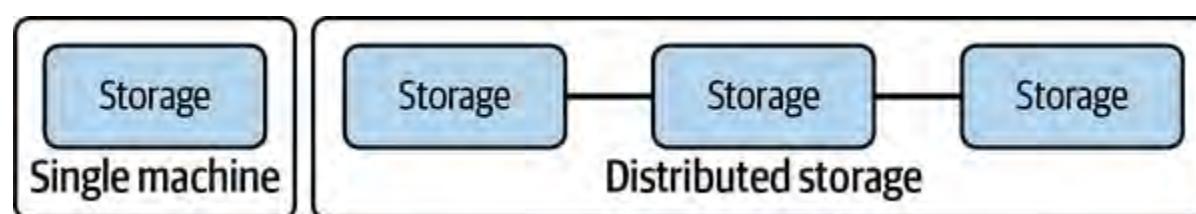
Table 6-1. A heuristic cache hierarchy displaying storage types with approximate pricing and performance characteristics

Storage type	Data fetch latency ^a	Bandwidth	Price
CPU cache	1 nanosecond	1 TB/s	N/A
RAM	0.1 microseconds	100 GB/s	\$10/GB
SSD	0.1 milliseconds	4 GB/s	\$0.20/GB
HDD	4 milliseconds	300 MB/s	\$0.03/GB
Object storage	100 milliseconds	10 GB/s	\$0.02/GB per month
Archival storage	12 hours	Same as object storage once data is available	\$0.004/GB per month

Data Storage Systems

Single Machine Versus Distributed Storage

- As data storage and access patterns become more complex and outgrow the usefulness of a single server
 - distributing data to more than one server becomes necessary
- Data can be stored on multiple servers, known as distributed storage
 - with purpose is to store data in a distributed fashion
- Distributed storage coordinates the activities of multiple servers
 - to store, retrieve, and process data faster and at a larger scale, all while providing redundancy in case a server becomes unavailable
 - provides built-in redundancy and scalability for large amounts of data
- Examples, object storage, Apache Spark, and cloud data warehouses rely on distributed storage architectures



Data Storage Systems(2)

File Storage

- A file is a data entity with specific read, write, and reference characteristics used by software and operating systems
- A file
 - has finite length - stream of bytes
 - supports append operations - append bytes to the files
 - supports random access - read from any location in the file or write updates to any location
- File storage systems organize files into a directory tree
 - /Users/Pravin/file.txt
- The filesystem stores each directory as metadata about the files and directories that it contains
 - consists of the name of each entity, relevant permission details, and a pointer to the actual entity
 - To find a file on disk, the operating system looks at the metadata at each hierarchy level and follows the pointer to the next subdirectory entity until finally reaching the file itself.

Data Storage Systems(3)

File Storage - Types

- Local disk storage
 - The most familiar type of file storage is an operating system–managed filesystem on a local disk partition of SSD or magnetic disk
 - New Technology File System (NTFS) and ext4 are popular filesystems on Windows and Linux respectively
 - OS handles the details of storing directory entities, files, and metadata
 - designed to write data to allow for easy recovery in the event of power loss during a write, though any unwritten data will still be lost.
 - support full read after write consistency; reading immediately after a write will return the written data
 - employs various locking strategies to manage concurrent writing attempts to a file
- Network-attached storage
 - provide a file storage system to clients over a network
 - prevalent solution for servers; they quite often ship with built-in dedicated NAS interface hardware
 - has performance penalties to accessing the filesystem over a network
 - significant advantages includes redundancy and reliability, fine-grained control of resources and file sharing across multiple machines
- Cloud filesystem services
 - provide a fully managed filesystem for use with multiple cloud VMs and applications, potentially including clients outside the cloud environment
 - should not be confused with standard storage attached to VMs—generally, block storage with a filesystem managed by the VM operating system
 - behave much like NAS solutions, but the details of networking, managing disk clusters, failures, and configuration are fully handled by the cloud vendor
 - Amazon Elastic File System (EFS) is an extremely popular example of a cloud filesystem service

Data Storage Systems(4)

Block Storage

- Fundamentally type of raw storage provided by SSDs and magnetic disks
- In the cloud, virtualized block storage is the standard for VMs
 - allow fine control of storage size, scalability, and data durability beyond that offered by raw disks.
- A block is the smallest addressable unit of data supported by a disk
 - was often 512 bytes of usable data on older disks, but it has now grown to 4,096 bytes for most current disks
 - also contain extra bits for error detection/correction and other metadata
- Blocks on magnetic disks are geometrically arranged on a physical platter
 - Two blocks on the same track can be read without moving the head, while reading two blocks on separate tracks requires a seek
 - Seek time can occur between blocks on an SSD, but this is infinitesimal compared to the seek time for magnetic disk tracks.
- Block storage applications
 - Transactional database systems generally access disks at a block level to lay out data for optimal performance
 - Operating system boot disks on cloud VMs

Data Storage Systems(5)

Block Storage - Types

- RAID
 - redundant array of independent disks
 - simultaneously controls multiple disks to
 - improve data durability
 - enhance performance
 - combine capacity from multiple drives
 - an array can appear to the operating system as a single block device
- Storage area network (SAN)
 - provide virtualized block storage devices over a network, typically from a storage pool
 - allow fine-grained storage scaling and enhance performance, availability, and durability
 - common with on-premises storage systems

Data Storage Systems(6)

Block Storage - Types

- Cloud virtualized block storage
 - similar to SAN but free engineers from dealing with SAN clusters and networking details
 - Example, **Amazon Elastic Block Store (EBS)** - default storage for Amazon EC2 virtual machines
 - store data separate from the instance host server but in the same zone to support high performance and low latency
 - allows EBS volumes to persist when an EC2 instance shuts down, when a host server fails, or even when the instance is deleted
 - suitable for applications such as databases, where data durability is a high priority
 - replicates all data to at least two separate host machines, protecting data if a disk fails
- Local instance volumes
 - Cloud providers also offer block storage volumes that are physically attached to the host server running a virtual machine
 - generally very low cost (included with the price of the VM in the case of **Amazon's EC2 instance store**) and provide low latency
 - when a VM shuts down or is deleted, the contents of the locally attached disk are lost, whether or not this event was caused by intentional user action
 - ensures that a new virtual machine cannot read disk contents belonging to a different customer

Data Storage Systems(7)

Object Storage

- Object Storage
 - contains objects of all shapes and sizes - a specialized file-like construct —TXT, CSV, JSON, images, videos, or audio
 - have grown in importance and popularity with the rise of big data and the cloud
 - Amazon S3, Azure Blob Storage, and Google Cloud Storage (GCS) are widely used
 - many cloud data warehouses (and a growing number of databases) utilize object storage as their storage layer
 - cloud data lakes generally sit on object stores
- Object storage was arguably one of the first “serverless” services
 - engineers don’t need to consider the characteristics of underlying server clusters or disks
- An object store is a key-value store for immutable data objects.
 - don’t support random writes or append operations; instead, they are written once as a stream of bytes
 - After this initial write, objects become immutable
 - To change data in an object or append data to it, must rewrite the full object
 - support random reads through range requests

Data Storage Systems(8)

Cache and Memory-Based Storage Systems

- RAM offers excellent latency and transfer speeds but extremely vulnerable to data loss
- RAM-based storage systems are generally focused on caching applications, presenting data for quick access and high bandwidth
 - Data should generally be written to a more durable medium for retention purposes
 - These ultra-fast cache systems are useful when data engineers need to serve data with ultra-fast retrieval latency
- Memcached and lightweight object caching
 - a key-value store designed for caching database query results, API call responses, and more
 - uses simple data structures, supporting either string or integer types
 - deliver results with very low latency while also taking the load off backend systems
- Redis, memory caching with optional persistence
 - a key-value store, but it supports somewhat more complex data types (such as lists or sets)
 - with a typical configuration, Redis writes data roughly every two seconds
 - Redis is thus suitable for extremely high-performance applications but can tolerate a small amount of data loss

Data Storage Systems(9)

The Hadoop Distributed File System

- HDFS is
 - based on Google File System (GFS) and was initially engineered to process data with the MapReduce programming model
 - similar to object storage but with a key difference: Hadoop combines compute and storage on the same nodes, where object stores typically have limited support for internal processing
- Management
 - breaks large files into blocks, chunks of data less than a few hundred megabytes in size
 - managed by the NameNode, which maintains directories, file metadata, and a detailed catalog describing the location of file blocks in the cluster
 - each block of data is replicated to three nodes - increases both the durability and availability of data
- Claims that Hadoop is dead! This is only partially true!
 - no longer a hot, bleeding-edge technology
 - Many Hadoop ecosystem tools such as Apache Pig are now on life support and primarily used to run legacy jobs
 - The pure MapReduce programming model has fallen by the wayside
- but HDFS remains widely used in various applications and organizations
 - HDFS is a key ingredient of many current big data engines, such as Amazon EMR
 - In fact, Apache Spark is still commonly run on HDFS clusters

Data Storage Systems(10)

Streaming Storage

- Streaming data has different storage requirements than nonstreaming data - **retention, replay!**
- Retention
 - In the case of message queues, stored data is temporal and expected to disappear after a certain duration
 - Distributed, scalable streaming frameworks like Apache Kafka now allow extremely long-duration streaming data retention
 - Kafka competitors (including Amazon Kinesis, Apache Pulsar, and Google Cloud Pub/Sub) also support long data retention
- Replay
 - allows a streaming system to return a range of historical stored data
 - standard data-retrieval mechanism for streaming storage systems
 - can be used to run batch queries over a time range or to reprocess data in a streaming pipeline

Data Storage Abstractions

- Data engineering storage abstractions are **data organization and query patterns** that sit at the heart of the data engineering lifecycle
 - are built atop the data storage systems
- The main types of abstractions are those that support **data science, analytics, and reporting** use cases
 - Data warehouse
 - Data lake
 - Data lakehouse
 - Data platforms
 - and data catalogs



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Data Science Infrastructure

Pravin Y Pawar

Adapted from
Effective Data Science Infrastructure
By Ville Tuulos

Paradigm Shift

- The biggest paradigm shifts in computing happen not when impossible things become possible, but when **possible things become easy**.
- Call it machine learning, artificial intelligence, or data science
 - is on the cusp of becoming radically **more approachable** than ever before:
 - More people will be able to build more, smarter applications
 - **This is super exciting!**

Lifecycle of a Data Science Project



Lifecycle of a data science project

Lifecycle of a Data Science Project

Common Elements

- Details of the **life cycle will naturally vary between companies and projects**:
 - How you develop a predictive model for customer lifetime value differs greatly from building self-driving cars.
- However, all data science and machine learning projects have a **few key elements in common**:
 - In the technical point of view, all projects **involve data and computation** at their foundation.
 - All projects will eventually need to address the question of **integrating results into production systems** which typically involves a great deal of software engineering.
 - Finally, in the human point of view, all projects involve **experimentation and iteration**, which many consider to be the central activity of data science.
- Certainly possible for individuals companies or teams to come up with their own bespoke processes and practices to conduct data science projects,
 - **a common infrastructure** can help to increase the number of projects that can be
 - executed simultaneously (**volume**),
 - speed up the time-to-market (**velocity**),
 - ensure that the results are robust (**validity**), and
 - make it possible to support a larger **variety** of projects.

Data Science Infrastructure

Evolution

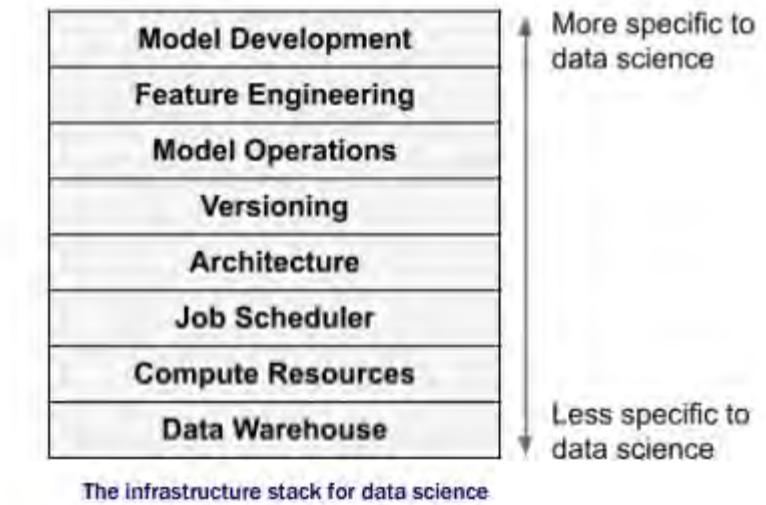
- The infrastructure for data science is going through an evolution
 - Primordial machine learning and optimization **libraries have existed for decades** without much other infrastructure
 - As of 2020, experiencing a Cambrian **explosion of data science libraries, frameworks, and infrastructures**
 - often driven by commercial interests, similar to what happened during the Dotcom boom.
- Widely shared patterns will emerge from this soup which will form the basis of a **common, open-source infrastructure stack for data science.**

Data Science Infrastructure (2)

The stack

- Because of culture of open source and relatively free technical information sharing between companies,
 - been able to observe and collect common patterns in data science projects and infrastructure components
 - implementation details vary, the major infrastructural layers are relatively uniform across a large number of projects
 - each layer can be implemented in various ways, driven by the specific needs of their environment and use cases
 - but the big picture is remarkably consistent!

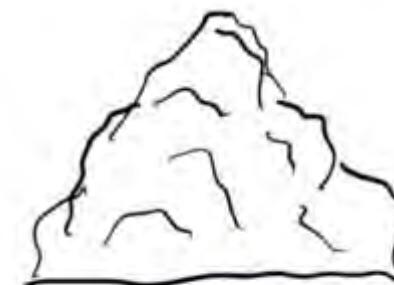
- The most fundamental, generic components
 - are at the bottom of the stack
- The layers become more specific to data science
 - towards the top of the stack



Data Science Infrastructure (3)

Components : Data Warehouse & Compute Resources

- Data Warehouse
 - Place where **data is stored**
 - Doesn't do anything by itself but **how data is stored is critical**
 - A key challenge is to **lay out the data** in a way that makes it **easily discoverable** and **efficiently accessible** by data science applications while maintaining a **high degree of durability**
 - naturally the data warehouse must not lose data.



Data Warehouse

- Compute Resources
 - Need to have a **way to do something with data**, in other words, take some data, apply an algorithm and get results
 - A challenge is that **algorithms come in many shapes and sizes** and need to **run many of them, often in parallel, very frequently**
 - Need a system that can provide computing **resources on demand**
 - The compute layer doesn't dictate what is computed
 - blindly accepts an algorithm from the outside world and executes it, consuming and producing data



Compute Resources

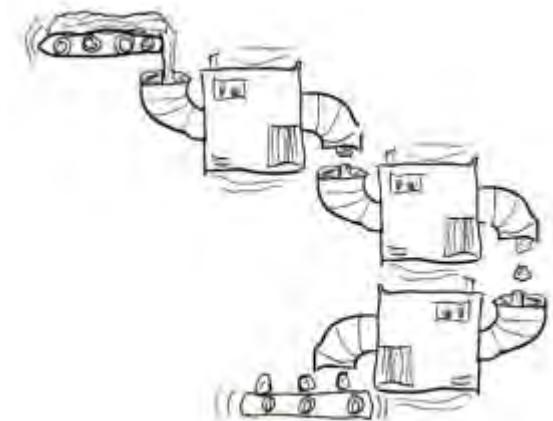
Data Science Infrastructure (4)

Components: Job Scheduler & Architecture

- Job Scheduler
 - Training a machine learning model or other data science application is rarely a one-time operation
 - More often it is a repeating process
 - Need to feed fresh data in the algorithm at a regular cadence
 - Job of the scheduling layer to keep all the data flowing and all the machines humming uninterrupted
 - A key challenge is to keep the scheduling system itself up all the time and make sure it can manage almost any number of concurrent machines and data pipelines
- Architecture
 - Data warehouse, compute resources, and job scheduler, are generic in nature: can be used to execute any workloads
 - At the architecture layer defines what application is and how it is going to work
 - includes defining an algorithm or several algorithms and how the scheduling layer is supposed to connect the algorithms and the data pipelines together
 - A key challenge is that of software engineering in general:
 - How to map a real-world problem into an effective and robust software architecture.



Job Scheduler

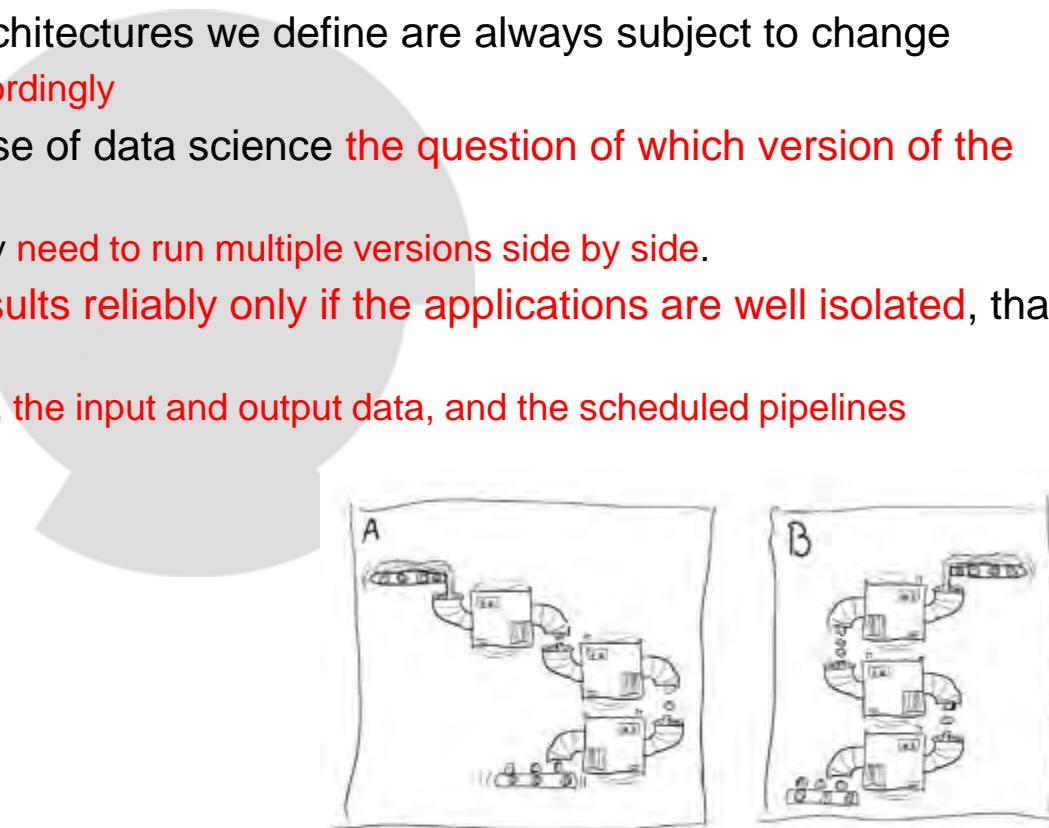


Architecture

Data Science Infrastructure (5)

Components: Versioning

- Versioning
 - “experimentation and iteration” as one of the common themes in data science projects
 - A concrete ramification of this is that architectures we define are always subject to change
 - makes sense to organize the work accordingly
 - To complicate matters further, in the case of data science **the question of which version of the application is better is often non-trivial**
 - To judge the versions properly, you may **need to run multiple versions side by side.**
 - A key **challenge** is that can **evaluate results reliably** only if the applications are **well isolated**, that is, they must not interfere with each other
 - Must isolate the algorithms themselves, the input and output data, and the scheduled pipelines

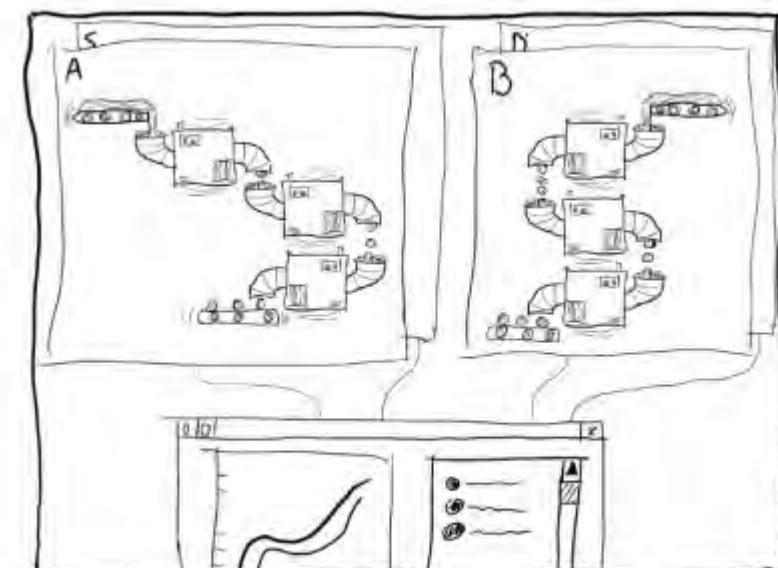


Versioning

Data Science Infrastructure (6)

Components: Model Operations

- Model Operations
 - Imagine four versions of an application running, each with their own version of code and data
 - Each version may be built by a separate data scientist, all of whom are all iterating on their versions independently.
 - The result, unless organized properly, can be a logistical nightmare!
 - Role of the model operations layer to help
 - to deploy, monitor, and assure validity of all models at all times,
 - without hindering the speed of experimentation
 - To make this possible, the infrastructure needs to track metadata about all executions and models, from prototype to production



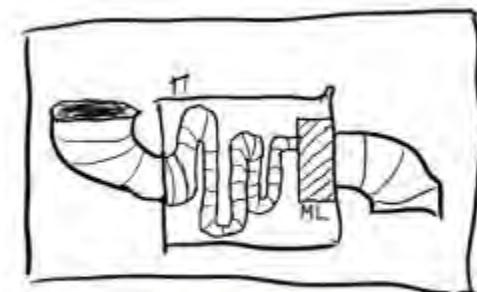
Model Operations

Data Science Infrastructure (7)

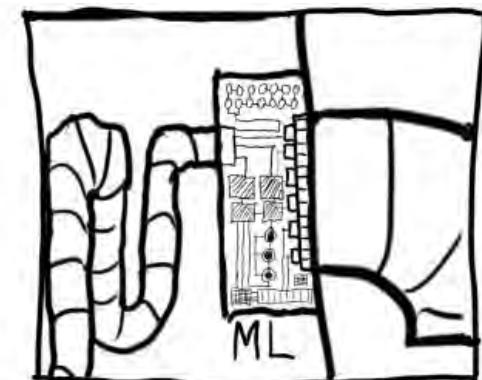
Components: Feature Engineering & Model Development

- Feature Engineering
 - Typically, the raw input data is not fed directly into the model
 - **but it goes through a sequence of data transformations**
 - Designing this sequence is a major part of the data scientist's work
 - when they are developing a new model or trying to improve performance of an existing model
 - A key challenge is to make this process as **efficient as possible**

- Model Development
 - At the very top of the stack there's the layer of model development
 - finding and describing a mathematical model that transforms features into desired outputs
 - **A key challenge, and the key expertise of a data scientist, is to understand what modeling approach is most appropriate for the task at hand and how to implement it typically using off-the-shelf libraries.**
 - Occupies only a tiny part of the end-to-end machinery that makes an effective data science application
 - **shouldn't underestimate the effort that it takes to embed intelligence in the surrounding environment, which is what this stack is all about!**



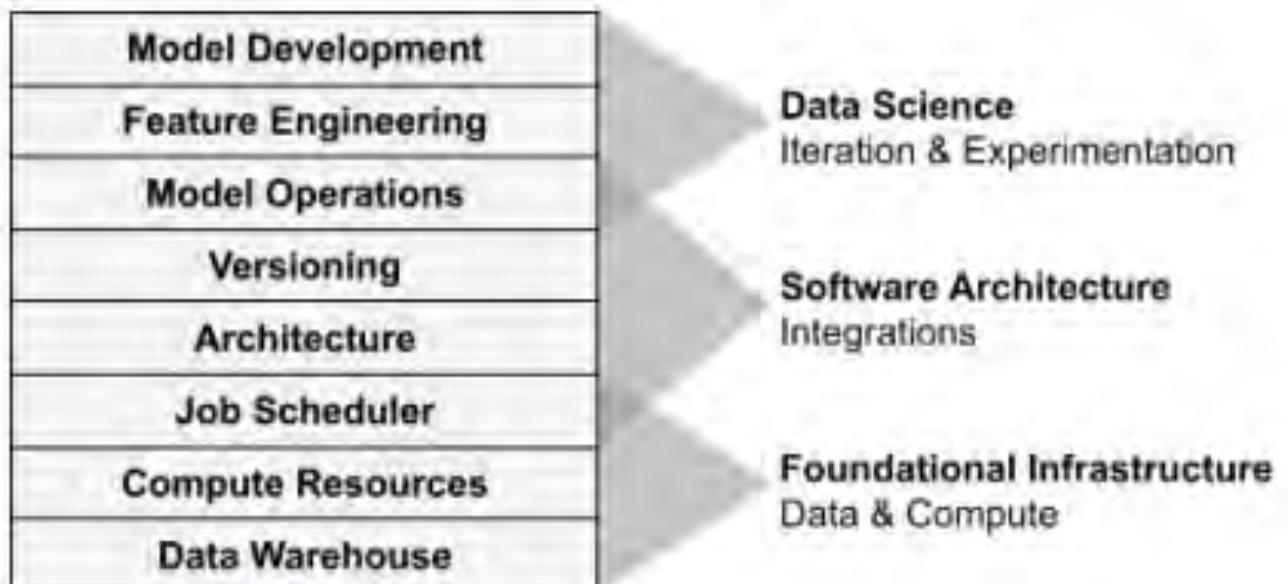
Feature Engineering



Model Development

Data Science Infrastructure (8)

Stack again



Concerns of a data science project mapped to the infrastructure layers

Data Science Infrastructure (9)

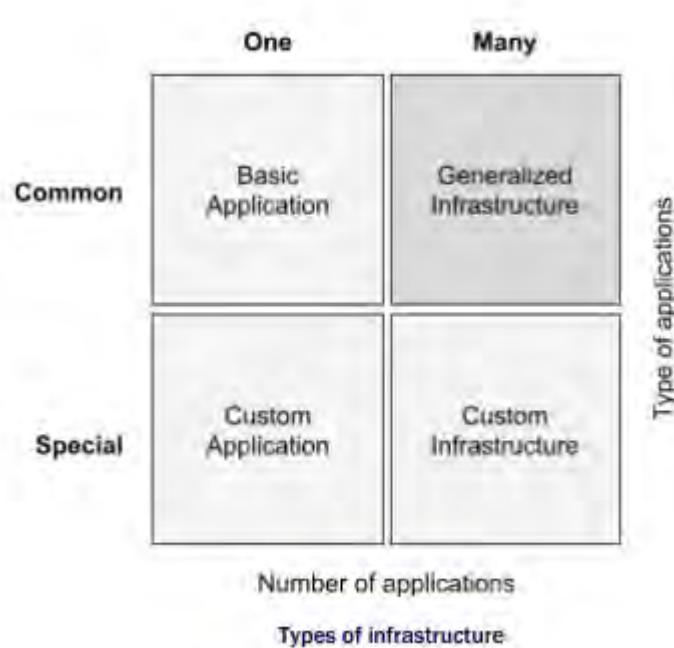
WHY THE STACK

- Each layer of the stack is a complex system of its own
 - meant to cover the needs of a typical data science project, end-to-end,
- Three common themes that are common to most data science projects
 - Every data science project regardless of the problem domain needs to deal with data and compute
 - so these layers form the foundational infrastructure
 - layers are agnostic of what exactly gets executed
 - The middle layers
 - define the software architecture of an individual data science application
 - What gets executed and how: the algorithms, data pipelines, deployment strategies, and how results should be distributed
 - Much about the work is about integrating existing software components together.
 - The top of the stack
 - realm of data science:
 - Defining a mathematical model and how to transform raw input to something that the model can process
 - can evolve quickly as the data scientist experiments with different approaches

Data Science Infrastructure (10)

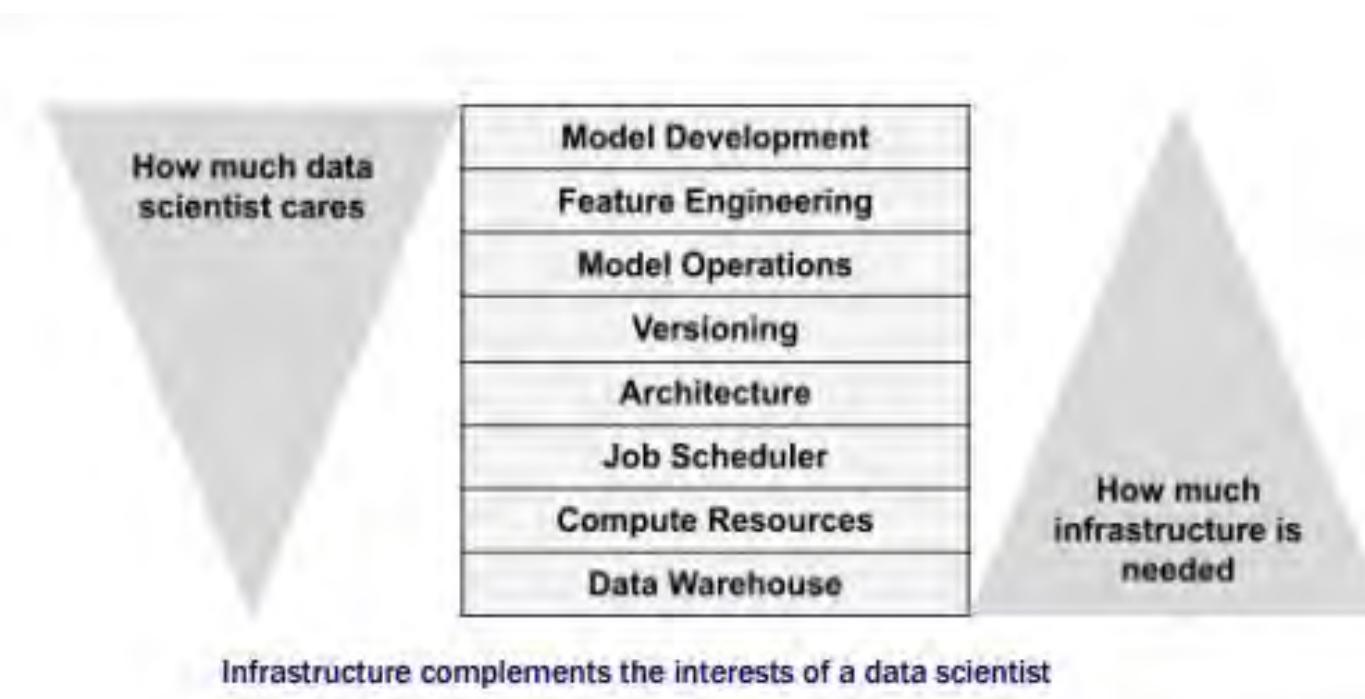
ONE SIZE DOESN'T FIT ALL

- What if your company needs a highly specialized data science application:
 - A self-driving car, a high-frequency trading system
 - Surely the infrastructure stack would need to look very different for such applications?



Data Science Infrastructure (11)

Data Scientists Freedom and Responsibility





Thank You!

In our next session:



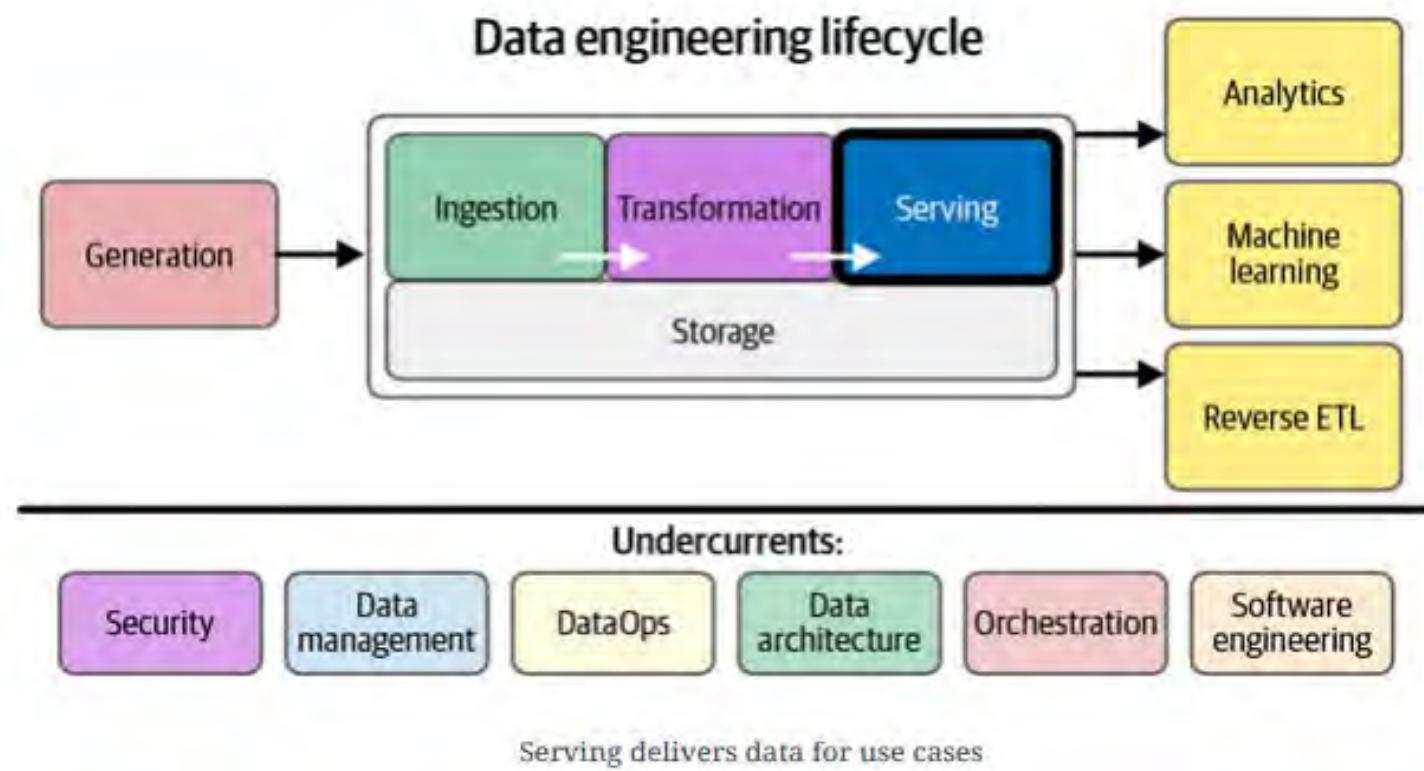
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Data Serving for Analytics and ML

Pravin Y Pawar

Adapted from
Fundamentals of Data Engineering
By Joe Reis and Matt Housley

Data Serving



Data Serving

Three Ways

- Serve data for analytics and BI
 - most traditional area of data serving
 - prepare data for use in statistical analysis, reporting, and dashboards
 - important as ever for stakeholders to have visibility into the business, organizational, and financial processes
- Serve data for ML applications
 - ML is not possible without high quality data, appropriately prepared
 - work with data scientists and ML engineers to acquire, transform, and deliver the data necessary for model training
- Serve data through reverse ETL
 - process of sending data back to data sources
 - For example, might acquire data from an adtech platform, run a statistical process on this data to determine cost-per-click bids, and then feed this data back into the ad tech platform

Analytics

- Discovering, exploring, identifying, and making visible key insights and patterns within data
 - Carried out using statistical methods, reporting, BI tools, and more
- Business Analytics
 - uses historical and current data to make strategic and actionable decisions
 - types of decisions tend to factor in longer-term trends and often involve a mix of statistical and trend analysis, alongside domain expertise and human judgment
 - typically falls into a few big areas—dashboards, reports, and ad hoc analysis
- Operational Analytics
 - uses data to take immediate action:
 - Operational analytics versus business analytics = immediate action versus actionable insights
 - big difference between operational and business analytics is **time**
 - as real-time updates can be impactful in addressing a problem when it occurs
- The line between business and operational analytics has begun to blur!
- Embedded Analytics
 - Whereas business and operational analytics are internally focused, a recent **trend is external-facing** or embedded analytics
 - With so much data powering applications, companies increasingly provide analytics to end users
 - Typically referred to as **data applications**, often with analytics dashboards embedded within the application itself
 - The landscape of embedded analytics is snowballing, and we expect that such data applications will become increasingly pervasive within the next few years.

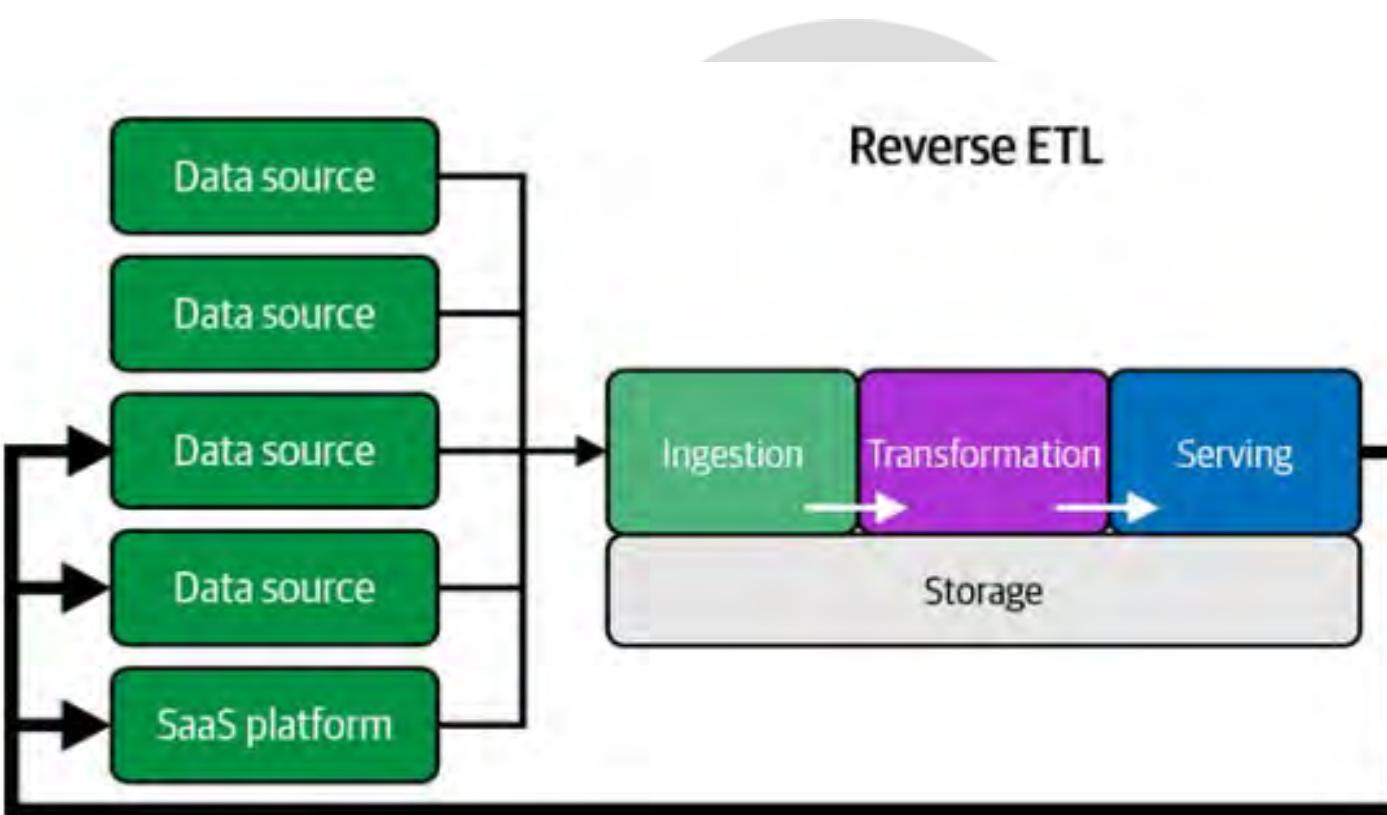
Machine Learning

- ML is increasingly common!
- ML engineering is rising (itself almost a parallel universe to data engineering)
 - Boundary between ML, data science, data engineering, and ML engineering is increasingly fuzzy, and this boundary varies dramatically between organizations
- In some organizations, ML engineers take over data processing for ML applications right after data collection or may even form an entirely separate and parallel data organization that handles the entire lifecycle for all ML applications.
 - Data engineers handle all data processing in other settings and then hand off data to ML engineers for model training
 - Data engineers may even handle some extremely ML-specific tasks, such as featurization of data

Reverse ETL

- Today, reverse ETL is a buzzword that describes serving data by loading it from an OLAP database back into a source system
 - increasingly recognized as a formal data engineering responsibility
- For example,
- A data engineer might pull customers and order data from a CRM and store it in a data warehouse
- This data is used to train a lead scoring model, whose results are returned to the data warehouse
- Company's sales team wants access to these scored leads to try to generate more sales
- Have a few options to get the results of this lead scoring model into the hands of the sales team
 - Can put the results in a dashboard for them to view
 - Or might email the results to them as an Excel file
- The challenge with these approaches is that they are not connected to the CRM, where a salesperson does their work
 - Why not just put the scored leads back into the CRM?
- Using reverse ETL and loading the scored leads back into the CRM is the easiest and best approach for this data product
- Reverse ETL takes processed data from the output side of the data engineering lifecycle and feeds it back into source systems

Reverse ETL(2)



Ways to Serve Data for Analytics and ML

File Exchange

- Is ubiquitous in data serving. We process data and generate files to pass to data consumers
 - A data scientist might load a text file (unstructured data) of customer messages to analyze the sentiments of customer complaints
 - A business unit might receive invoice data from a partner company as a collection of CSVs (structured data)
 - An analyst must perform some statistical analysis on these files
- The way files are served depends on several factors, such as these:
 - Use case — business analytics, operational analytics, embedded analytics
 - The data consumer's data-handling processes
 - The size and number of individual files in storage
 - Who is accessing this file
 - Data type — structured, semistructured, or unstructured

Ways to Serve Data for Analytics and ML(2)

Databases

- Critical layer in serving data for analytics and ML - serving data from OLAP databases (e.g., data warehouses and data lakes)
 - Serving data involves querying a database and then consuming those results for a use case
 - An analyst or data scientist might query a database by using a SQL editor and export those results to a CSV file for consumption by a downstream application, or analyze the results in a notebook (
- Carries a variety of benefits
 - A database imposes order and structure on the data through schema
 - Databases can offer fine-grained permission controls at the table, column, and row level, allowing database administrators to craft complex access policies for various roles
 - Databases can offer high serving performance for large, computationally intensive queries and high query concurrency
- A data scientist might connect to a database, extract data, and perform feature engineering and selection
 - Converted dataset is then fed into an ML model
 - offline model is trained and produces predictive results
 - Data engineers are quite often tasked with managing the database-serving layer.

Ways to Serve Data for Analytics and ML(3)

Streaming Systems

- Increasingly important in the realm of serving
 - type of serving may involve emitted metrics, which are different from traditional queries
- Operational analytics databases playing a growing role in this area
 - allow queries to run across a large range of historical data, encompassing up-to-the-second current data
 - combine aspects of OLAP databases with stream-processing systems
 - increasingly work with streaming systems to serve data for analytics and ML, so get familiar with this paradigm

Ways to Serve Data for Analytics and ML(4)

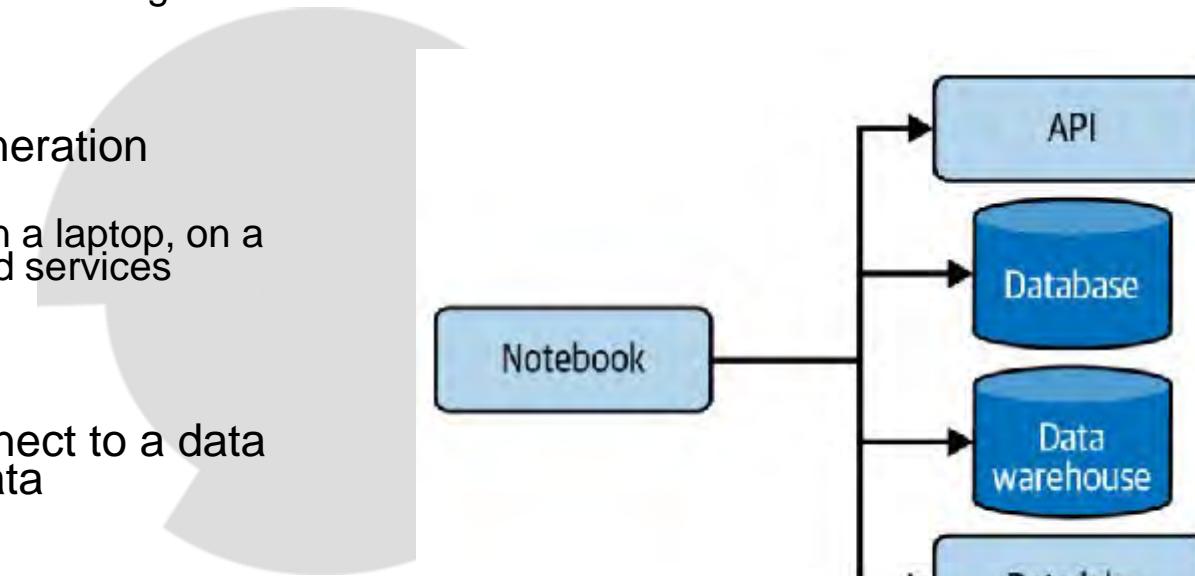
Query Federation

- Query federation pulls data from multiple sources, such as data lakes, RDBMSs, and data warehouses
 - becoming more popular as distributed query virtualization engines gain recognition as ways to serve queries without going through the trouble of centralizing data in an OLAP system
 - Trino and Presto and managed services such as Starburst
- End user might be querying several systems—OLTP, OLAP, APIs, filesystems, etc.
 - Instead of serving data from a single system, now serving data from multiple systems, each with its usage patterns, quirks, and nuances
 - If federated queries touch live production source systems, need to ensure that the federated query won't consume excessive resources in the source
- Federation allows ad hoc queries for performing exploratory analysis, blending data from various systems without the complexity of setting up data pipelines or ETL
 - Federated queries also provide read-only access to source systems, which is great when you don't want to serve files, database access, or data dump
 - The end user reads only the version of the data they're supposed to access and nothing more
 - Query federation is a great option to explore for situations where access and compliance are critical

Ways to Serve Data for Analytics and ML(5)

Serving Data in Notebooks

- Data scientists often use notebooks in their day-to-day work
 - for exploring data, engineering features, or training a model
- Jupyter Notebook, along with its next-generation iteration, JupyterLab
 - open source and can be hosted locally on a laptop, on a server, or through various cloud-managed services
 - Jupyter stands for Julia, Python, and R
- Data scientists will programmatically connect to a data source, such as an API, a database, a data warehouse, or a data lake
- In a notebook, all connections are created using
 - the appropriate built-in or imported libraries to load a file from a filepath
 - connect to an API endpoint
 - or make an ODBC connection to a database





Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Three Levels of ML Software

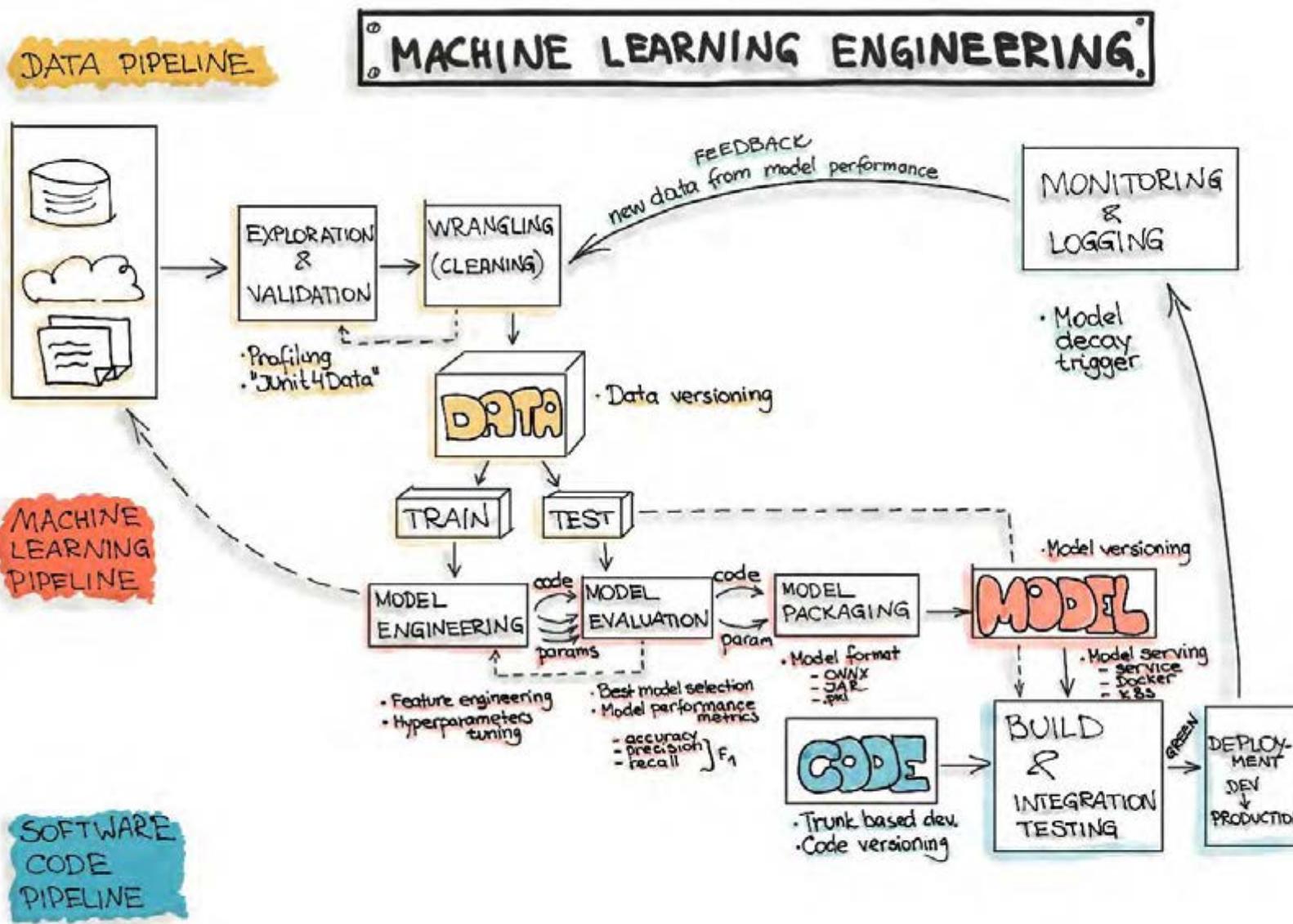
Pravin Y Pawar

Extracted from
[An Overview of the End-to-End Machine Learning Workflow](#)

Three levels of ML Software

- ML/AI is rapidly adopted by new applications and industries!
- Goal of a machine learning project is to build a statistical model by using collected data and applying machine learning algorithms
 - Yet building successful ML-based software projects is still difficult because
- Every ML-based software needs to manage three main assets:
 - Data
 - Model
 - and Code
- Essential technical methodologies - involved in the development of the Machine Learning-based software
 - **Data Engineering:** data acquisition & data preparation,
 - **ML Model Engineering:** ML model training & serving, and
 - **Code Engineering :** integrating ML model into the final product.

Machine Learning Engineering



Data Engineering

- The initial step in any data science workflow is to **acquire and prepare the data to be analyzed**
 - Typically, data is being integrated from **various resources** and has **different formats**
- Data Preparation
 - according to Gartner “**an iterative and agile process for exploring, combining, cleaning and transforming raw data into curated datasets for data integration, data science, data discovery and analytics/business intelligence (BI) use cases**”.
 - is reported to be the most expensive with respect to resources and time
- Data preparation is a critical activity in the data science workflow
 - important to avoid the propagation of data errors to the next phase, data analysis
 - would result in the derivation of wrong insights from the data

Data Engineering(2)

Sequence of operations on the data leading to training and testing datasets for ML algorithms

- Data Ingestion
 - Collecting data by using various frameworks and formats, such as Spark, HDFS, CSV, etc
 - Might also include synthetic data generation or data enrichment.
- Data Exploration
 - Includes data profiling to obtain information about the content and structure of the data
 - The output of this step is a set of metadata, such as max, min, avg of values
- Data Validation
 - Operations are user-defined error detection functions, which scan the dataset in order to spot some errors
- Data Wrangling (Cleaning)
 - The process of re-formatting particular attributes and correcting errors in data, such as missing values imputation
- Data Labeling
 - The operation of the Data Engineering pipeline, where each data point is assigned to a specific category
- Data Splitting
 - Splitting the data into training, validation, and test datasets to be used during the core machine learning stages to produce the ML model

Model Engineering

The core of the ML workflow

- Phase of writing and executing machine learning algorithms to obtain an ML model
- The Model Engineering pipeline includes a number of operations that lead to a final model:
 - Model Training
 - The process of applying the machine learning algorithm on training data to train an ML model
 - includes feature engineering and the hyperparameter tuning for the model training activity
 - Model Evaluation
 - Validating the trained model to ensure it meets original codified objectives before serving the ML model in production to the end-user
 - Model Testing
 - Performing the final “Model Acceptance Test” by using the hold backtest dataset
 - Model Packaging
 - The process of exporting the final ML model into a specific format (e.g. PMML, PFA, or ONNX)
 - which describes the model, in order to be consumed by the business application

Model Deployment

- Need to deploy model as part of a business application such as a mobile or desktop application
 - The ML models require various data points (feature vector) to produce predictions
 - The final stage of the ML workflow is the integration of the previously engineered ML model into existing software
- Includes the following operations:
- Model Serving
 - The process of addressing the ML model artifact in a production environment
- Model Performance Monitoring
 - The process of observing the ML model performance based on live and previously unseen data, such as prediction or recommendation
 - interested in ML-specific signals, such as prediction deviation from previous model performance
 - signals might be used as triggers for model re-training
- Model Performance Logging
 - Every inference request results in the log-record



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

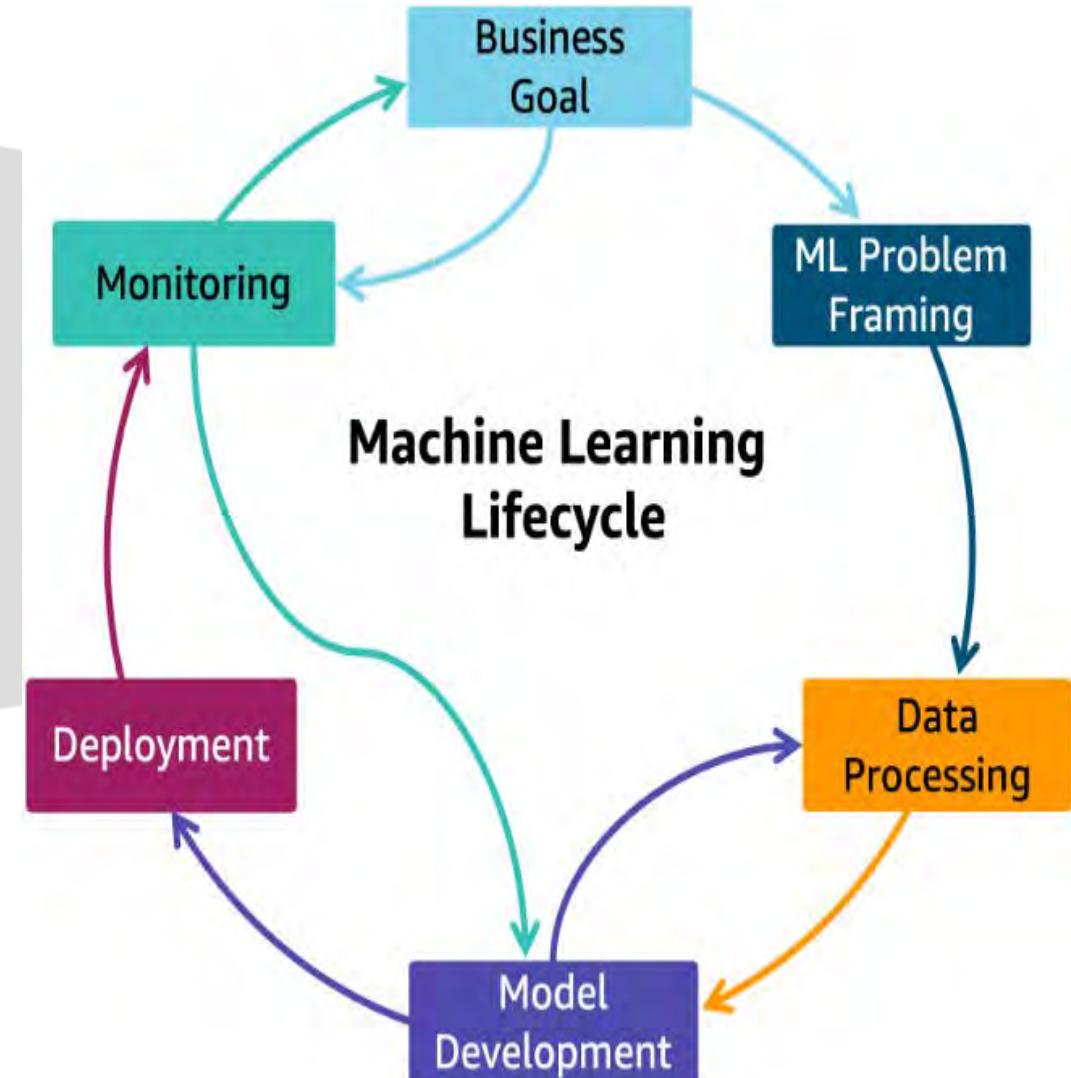
Machine Learning Lifecycle

Pravin Y Pawar

Adapted from AWS Well-Architected machine learning

Machine Learning Lifecycle

- Cyclic iterative process with instructions, and best practices to use across defined phases while developing an ML workload
 - adds clarity and structure for making a machine learning project successful
- The end-to-end machine learning lifecycle process includes the following phases:
 - Business goal identification
 - ML problem framing
 - Data processing (data collection, data preprocessing, feature engineering)
 - Model development (training, tuning, evaluation)
 - Model deployment (inference, prediction)
 - Model monitoring
- The phases of the ML lifecycle are not necessarily sequential in nature and can have feedback loops
 - to interrupt the cycle across the lifecycle phases



Machine Learning Lifecycle(2)

Phases

- Business goal
 - should have a clear idea of the problem, and the business value to be gained by solving that problem
 - must be able to measure business value against specific business objectives and success criteria
- ML problem framing
 - the business problem is framed as a machine learning problem
 - what is observed and what should be predicted (known as a label or target variable)
 - Determining what to predict and how performance and error metrics must be optimized is a key step in this phase
- Data processing
 - Training an accurate ML model requires data processing to convert data into a usable format
 - includes collecting data, preparing data
 - and feature engineering that is the process of creating, transforming, extracting, and selecting variables from data
- Model development
 - consists of model building, training, tuning, and evaluation
 - includes creating a CI/CD pipeline that automates the build, train and release to staging and production environments
- Deployment
 - After a model is trained, tuned, evaluated and validated, one can deploy the model into production
 - can then make predictions and inferences against the model
- Monitoring
 - ensures model is maintaining a desired level of performance through early detection and mitigation

ML lifecycle phase — Business goal

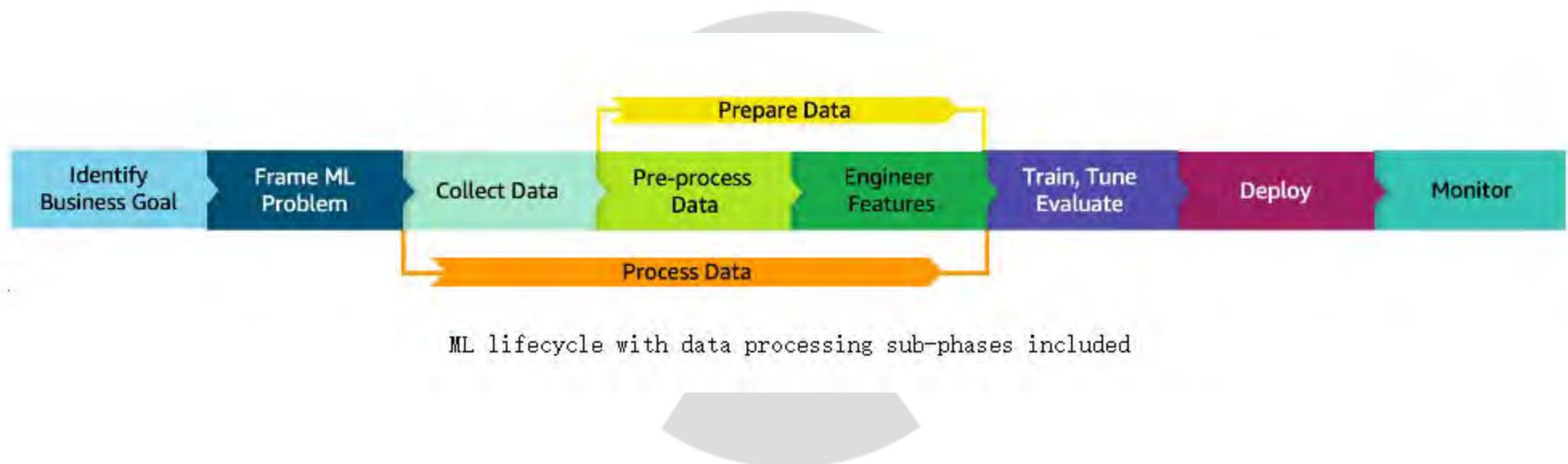
- The most important phase particularly challenging when considering ML solutions because ML is a constantly evolving technology!
- After success criteria is defined, evaluate organization's ability to move toward that target
 - The target should be **achievable** and provide a clear path to production
 - Involve
 - all relevant stakeholders from the beginning to align them to this target
 - any new business processes that will result from this initiative
- Steps in this phase:
 - Understand business requirements
 - Form a business question
 - Review a project's ML feasibility and data requirements
 - Evaluate the cost of data acquisition, training, inference, and wrong predictions
 - Review proven or published work in similar domains, if available
 - Determine key performance metrics, including acceptable errors
 - Define the machine learning task based on the business question
 - Identify critical, must have features
 - Design small, focused POCs to validate all of the preceding
 - Evaluate if bringing in external data sources will improve model performance
 - Establish pathways to production
 - Consider new business processes that may come out of this implementation
 - Align relevant stakeholders with this initiative

ML lifecycle phase — ML problem framing

- Business problem is framed as a machine learning problem
 - what is observed and what should be predicted (known as a label or target variable)
 - Determining what to predict and how performance must be optimized is a key step in ML
- Steps in this phase:
 - Define criteria for a successful outcome of the project
 - Establish an observable and quantifiable performance metric for the project, such as accuracy
 - Help ensure business stakeholders understand and agree with the defined performance metrics
 - Formulate the ML question in terms of inputs, desired outputs, and the performance metric to be optimized
 - Evaluate whether ML is the right approach
 - Some business problems don't need ML, simple business rules can do a much better job
 - For other business problems, there might not be sufficient data to apply ML as a solution
 - Create a strategy to achieve the data sourcing and data annotation objective
 - Start with a simple model that is easy to interpret, and which makes debugging more manageable

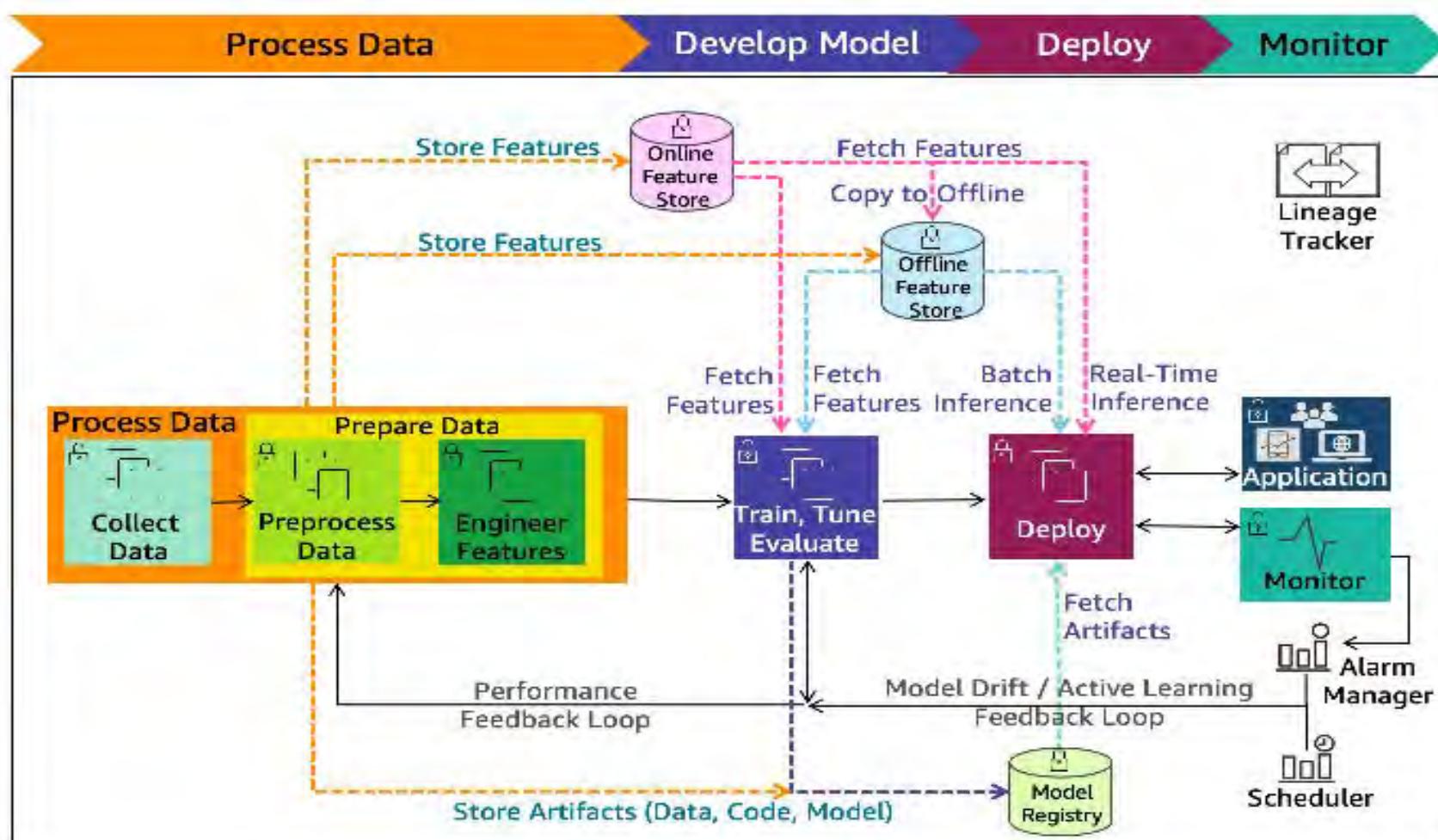
ML lifecycle architecture diagram

ML lifecycle with data processing sub-phases included



ML lifecycle architecture diagram(2)

ML lifecycle with detailed phases and expanded components



ML lifecycle with detailed phases and expanded components

ML lifecycle architecture diagram(3)

Components

- Online/Offline feature store
 - reduces duplication and rerun of feature engineering code across teams and projects
 - Online store with low-latency retrieval capabilities is ideal for real-time inference
 - Offline store should maintain a history of feature values and is suited for training and batch scoring
- Model registry
 - A repository for storing ML model artifacts including trained model and related metadata (data, code, model)
 - enables lineage for ML models as it can act as a version control system
- Performance feedback loop
 - Automates model performance evaluation tasks initiated from the model development to data processing phase
- Model drift feedback loop
 - Automates model update re-training tasks initiated from the production deployment to data processing phase

ML lifecycle architecture diagram(4)

Components

- Alarm manager
 - receives the alerts from the model monitoring system
 - runs actions by publishing notifications to services that can deliver alerts to target applications to handle them
 - The model update re-training pipeline is one such target application
- Scheduler
 - A scheduler can initiate a re-training at business defined intervals
- Lineage tracker
 - enables reproducible machine learning experiences
 - enables re-creating the ML environment at a specific point-in-time, reflecting the versions of all resources and environments at that time
 - collects references to traceable data, model and infrastructure resource changes

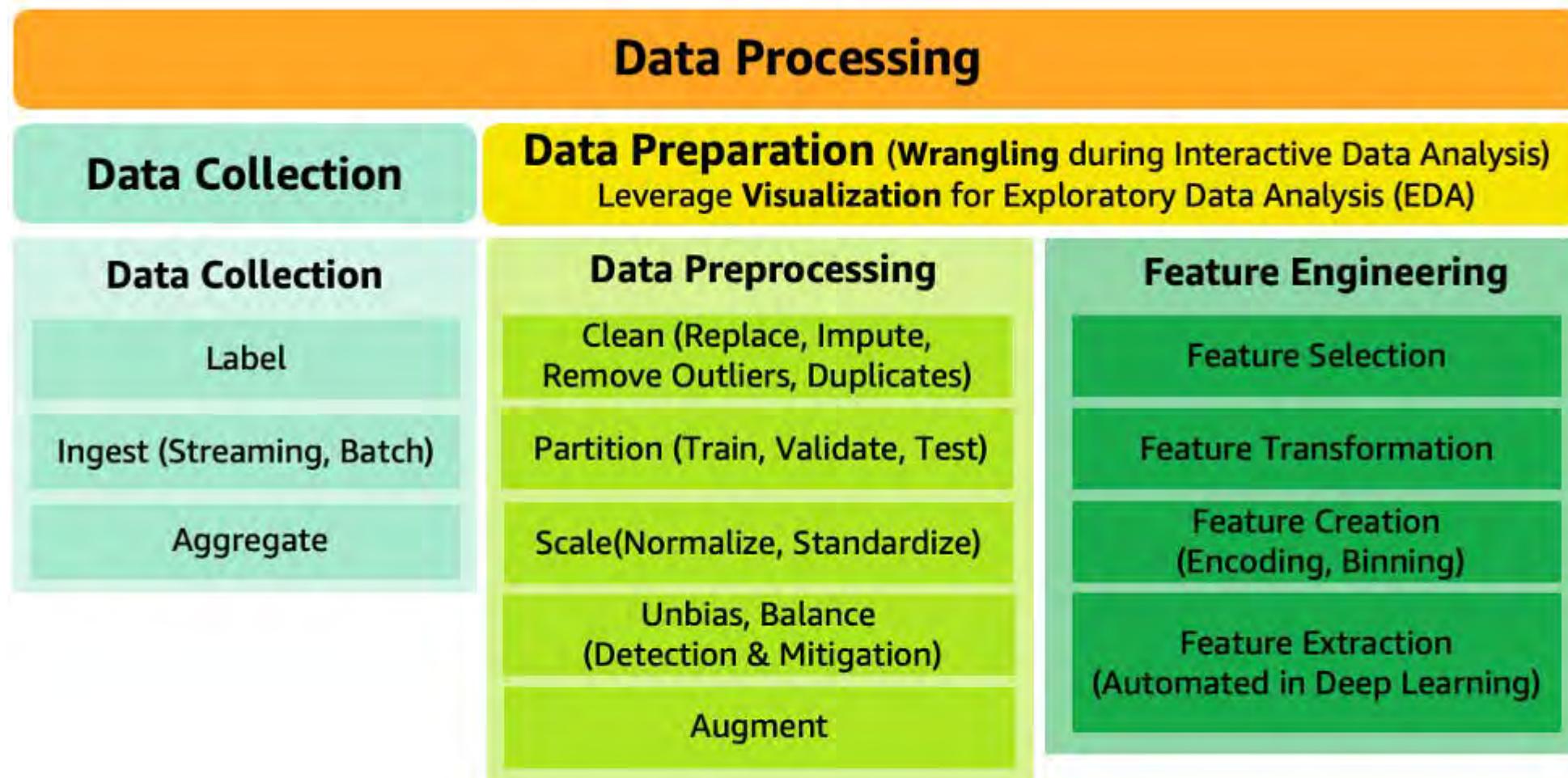
ML lifecycle phase - Data processing

Need

- In ML workloads, the data (inputs and corresponding desired output) serves important functions including:
 - Defining the goal of the system: the output representation and the relationship of each output to each input, by means of input/output pairs
 - Training the algorithm that associates inputs to outputs
 - Measuring the performance of the trained model, and evaluating whether the performance target was met
 - Building baselines to monitor the performance of the models deployed to production
- Data processing includes data collection and data preparation
 - Data preparation includes data preprocessing and feature engineering
 - Data wrangling is data preparation during the interactive data analysis and model development
 - Data Visualization can help with exploratory data analysis (EDA)
 - EDA can help with understanding data, sanity checks, and validating the quality of the data
- The same sequence of data processing steps that you apply to the training data is also applied to the inference requests.

ML lifecycle phase - Data processing(2)

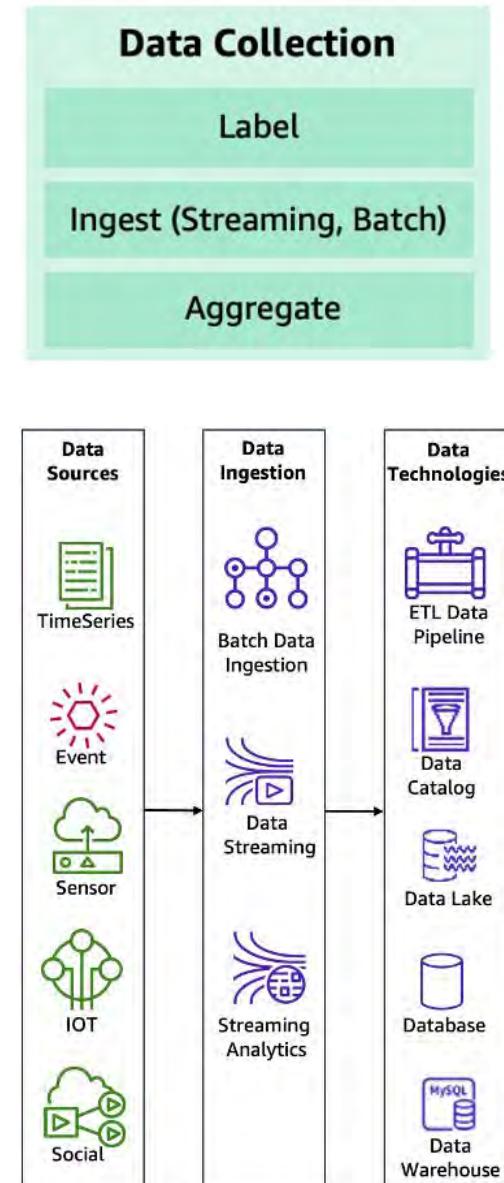
Data processing components



ML lifecycle phase - Data processing(3)

Data collection

- One of the first steps in the ML lifecycle is to identify what data is needed
 - Then evaluate the various means available for collecting that data to train model
- Activities in the data collection phase:
 - Label — Data for which already know the target answer is called labeled data
 - If labels are missing, then some effort is required to label it (manual or automated)
 - Ingest & Aggregate
 - Data collection includes ingesting and aggregating data from multiple data sources
- Some components of the ingest and aggregate:
 - Data sources
 - include time-series, events, sensors, IoT devices, and social networks, depending on the nature of the use case
 - Data ingestion
 - processes and technologies capture and store data on storage media
 - can occur in real-time using streaming technologies or historical mode using batch technologies
 - Data technologies
 - Data storage technologies vary from transactional (SQL) databases, to data lakes and data warehouses
 - ETL pipeline technology automates and orchestrates the data movement and transformations across cloud services and resources
 - A data lake technology enables storing and analyzing structured and unstructured data



ML lifecycle phase - Data processing(4)

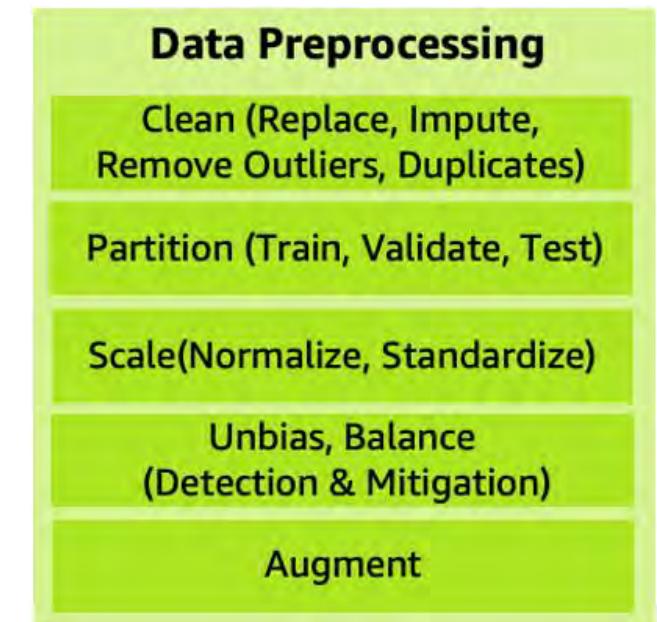
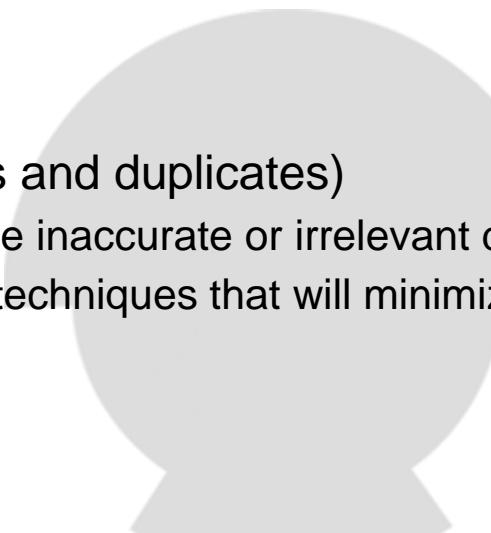
Data preparation

- ML models are only as good as the data that is used to train them
 - Ensure that suitable training data is available and is optimized for learning and generalization
- Data preparation includes
 - data preprocessing
 - feature engineering
- A key aspect to understanding data is to identify patterns which are often not evident with data in tables
 - Exploratory data analysis (EDA) with visualization tools can help quickly gain a deeper understanding of data
 - Prepare data using wrangler tools for interactive data analysis and model building
 - The no-code/low-code, automation, and visual capabilities improve the productivity and reduces the cost for interactive analysis

ML lifecycle phase - Data processing(5)

Data preprocessing

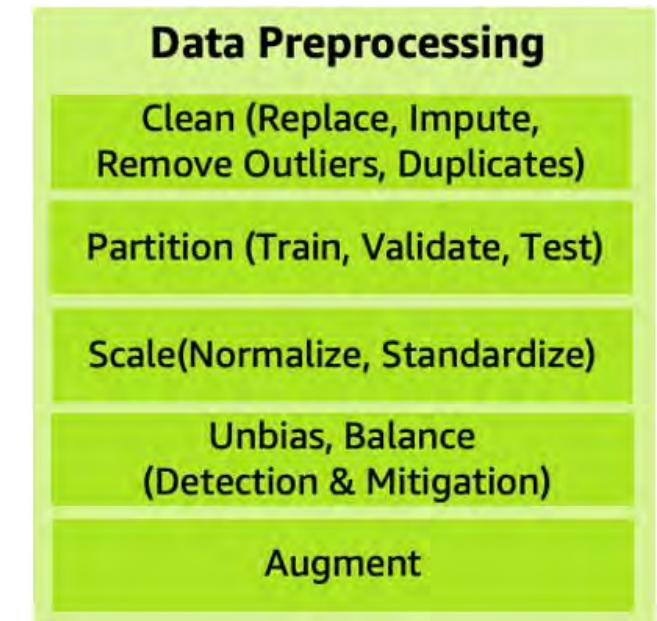
- Data preprocessing puts data into the right shape and quality for training
 - Many strategies: data cleaning, balancing, replacing, imputing, partitioning, scaling, augmenting and unbiasing
- Clean (replace, impute, remove outliers and duplicates)
 - Remove outliers and duplicates, replace inaccurate or irrelevant data
 - Correct missing data using imputation techniques that will minimize bias as part of data cleaning
- Partition
 - To prevent ML models from overfitting and evaluate trained model accurately, randomly split data into train, validate, and test sets
 - Care is needed to avoid data leakage
 - Data leakage happens when information from hold-out test dataset leaks into the training data
 - One way to avoid data leakage is to remove duplicates before splitting of the data



ML lifecycle phase - Data processing(6)

Data preprocessing

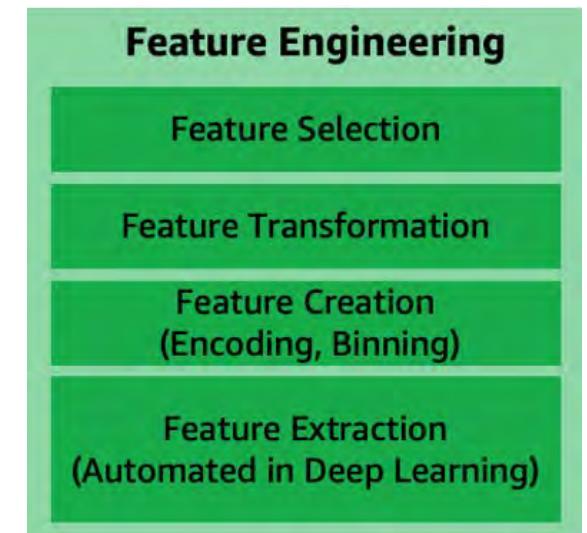
- Scale (normalize, standardize)
 - Having features on a similar scale close to normally distributed will ensure that each feature is equally important
 - make it easier for most ML algorithms including K-Means, KNN, PCA, gradient descent
 - Standardization better handles the outliers
- Unbias, balance (detection & mitigation)
 - Detecting and mitigating bias will help avoiding inaccurate model results
 - Biases are imbalances in the accuracy of predictions across different groups, such as age or income bracket
 - Biases can come from the data or algorithm used to train your model
- Augment
 - increases the amount of data artificially by synthesizing new data from existing data
 - can help regularize and reduce overfitting



ML lifecycle phase - Data processing(7)

Feature engineering

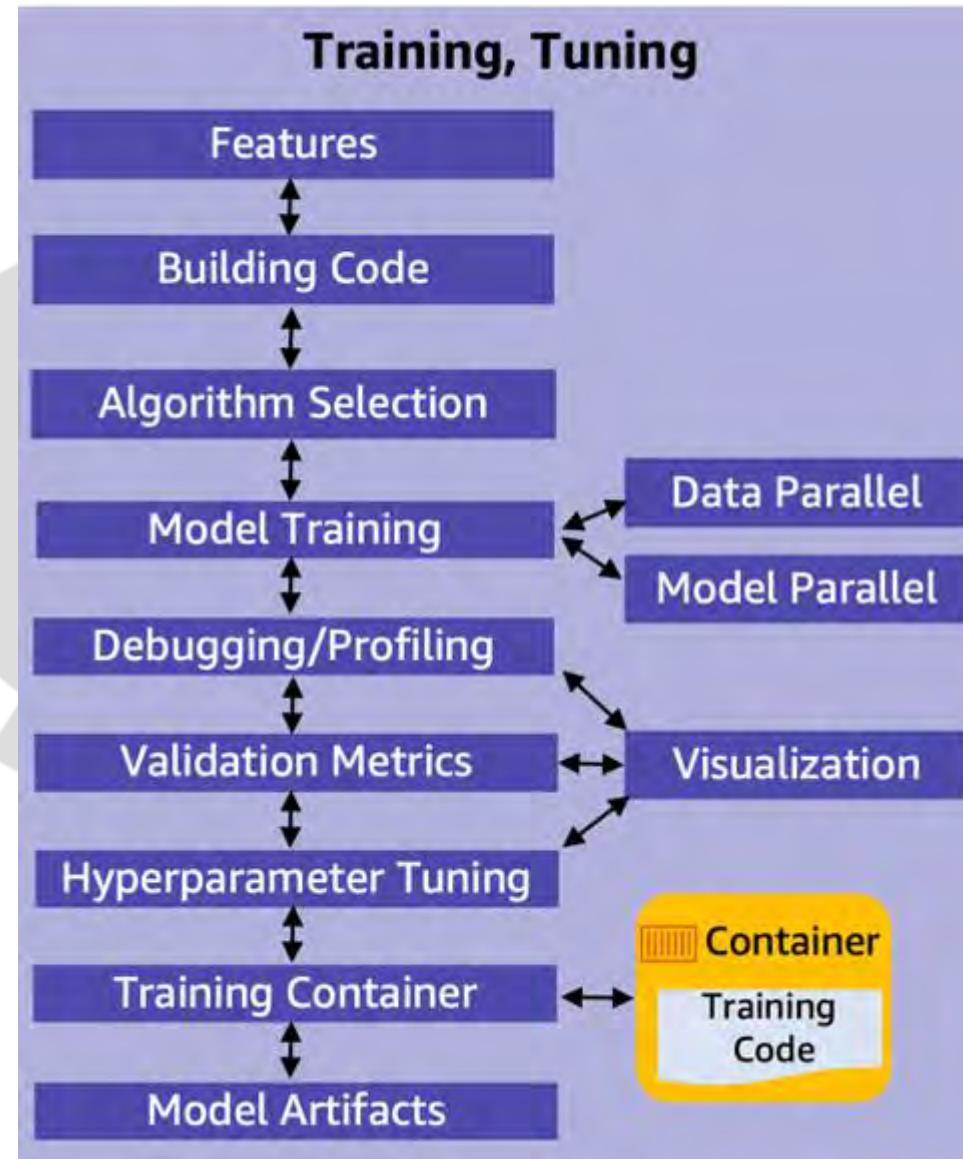
- Every unique attribute of the data is considered a feature
- Feature engineering is a process to select and transform variables when creating a predictive model using machine learning or statistical modeling
 - includes feature creation, feature transformation, feature extraction, and feature selection
 - With deep learning, the feature engineering is automated as part of the algorithm learning
- Feature creation
 - is creating new features from existing data to help with better predictions
 - Examples of feature creation techniques include: one-hot-encoding, binning, splitting, and calculated features
- Feature transformation and imputation
 - manage replacing missing features or features that are not valid
 - Some techniques include: forming Cartesian products of features, non-linear transformations (such as binning numeric variables into categories), and creating domain-specific features
- Feature extraction
 - involves reducing the amount of data to be processed using dimensionality reduction techniques
 - reduces the amount of memory and computing power required, while still accurately maintaining original data characteristics
 - Techniques include: PCA, ICA, and LDA
- Feature selection
 - process of selecting subset of extracted features which is relevant and contributes to minimizing the error rate of a trained model
 - Feature importance score and correlation matrix can be factors in selecting the most relevant features for model training



ML lifecycle phase – Model development

Model building, training, tuning

- In this phase, select a machine learning algorithm that is appropriate for problem and then train the ML model
 - provide the algorithm with the training data
 - set an objective metric for the ML model to optimize on
 - and set the hyperparameters to optimize the training process



ML lifecycle phase – Model development(2)

Model training, tuning activities

- Model training, tuning, and evaluation require prepared data and engineered features
- Features selection
 - Features are selected as part of the data processing after a bias strategy is implemented
- Building code
 - Model development includes building the algorithm and its supporting code
 - should support version control, and continuous build, test, and integration through a pipeline
- Algorithm selection
 - Selecting the right algorithm involves running many experiments with parameter tunings across available options
 - Factors to consider when evaluating each option can include **accuracy, explainability, training/prediction time, memory requirements**
- Model training (data training parallel, model training parallel)
 - The process of training an ML model involves providing an ML algorithm with training data to learn from
 - Distributed training enables splitting large models and training datasets across computing instances to reduce runtime to fraction of it takes to do manually
 - Model parallelism is the process of splitting a model up between multiple devices or nodes.
 - Data parallelism is the process of splitting the training set in mini-batches evenly distributed across nodes
 - each node only trains the model on a fraction of the total dataset

ML lifecycle phase – Model development(3)

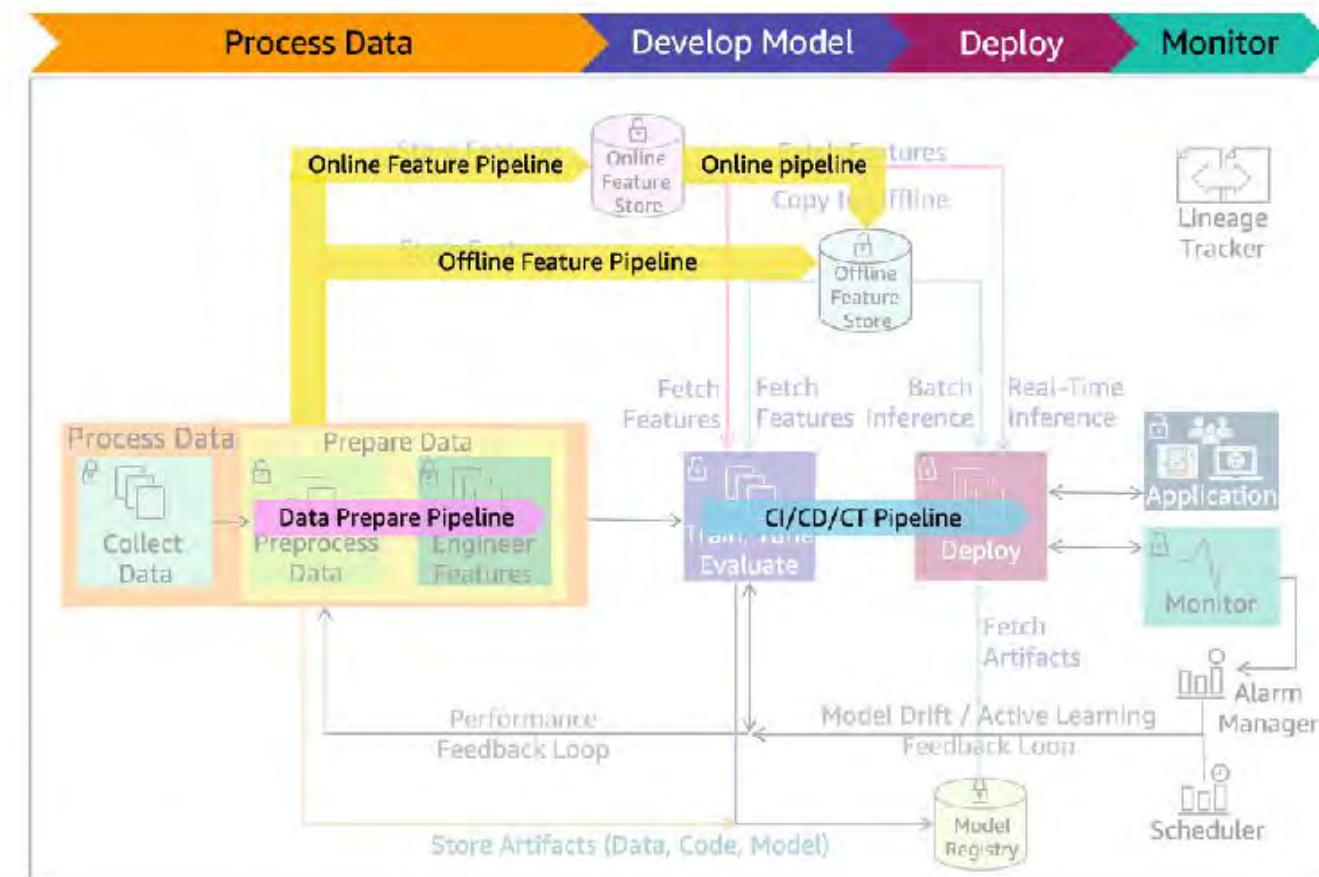
Model training, tuning activities

- Debugging/profiling
 - A machine learning training job can have problems including:
 - system bottlenecks, overfitting, saturated activation functions, and vanishing gradients. These problems can compromise model performance
 - A debugger provides visibility into the ML training process through monitoring, recording, and analyzing data
 - captures the state of a training job at periodic intervals
- Validation metrics
 - Typically, a training algorithm computes several metrics, such as loss and prediction accuracy
 - determine if the model is learning and generalizing well for making predictions on unseen data
 - Metrics reported by the algorithm depend on the business problem and the ML technique used
- Hyperparameter tuning
 - Settings that can be tuned to control the behavior of the ML algorithm are referred to as hyperparameters
 - The number and type of hyperparameters in ML algorithms are specific to each model
 - Examples of commonly used hyperparameters include: learning rate, number of epochs, hidden layers, hidden units, and activation functions
 - Hyperparameter tuning, or optimization, is the process of choosing the optimal hyperparameters for a learning algorithm
- Training code container
 - Create container images with training code and its entire dependency stack
 - will enable training machine learning algorithms and deploy models quickly and reliably at any scale
- Model artifacts
 - Model artifacts are the output that results from training a model
 - typically consist of trained parameters, a model definition that describes how to compute inferences, and other metadata.
- Visualization
 - Enables exploring and understanding data during metrics validation, debugging, profiling, and hyperparameter tuning.

ML lifecycle phase – Model development(4)

ML lifecycle with pre-production pipelines

- The data prepare pipeline automates data preparation tasks
- The feature pipeline automates the storing, fetching, and copying of the features into and from online/offline store
- The CI/CD/CT pipeline automates the build, train, and release to staging and production environments



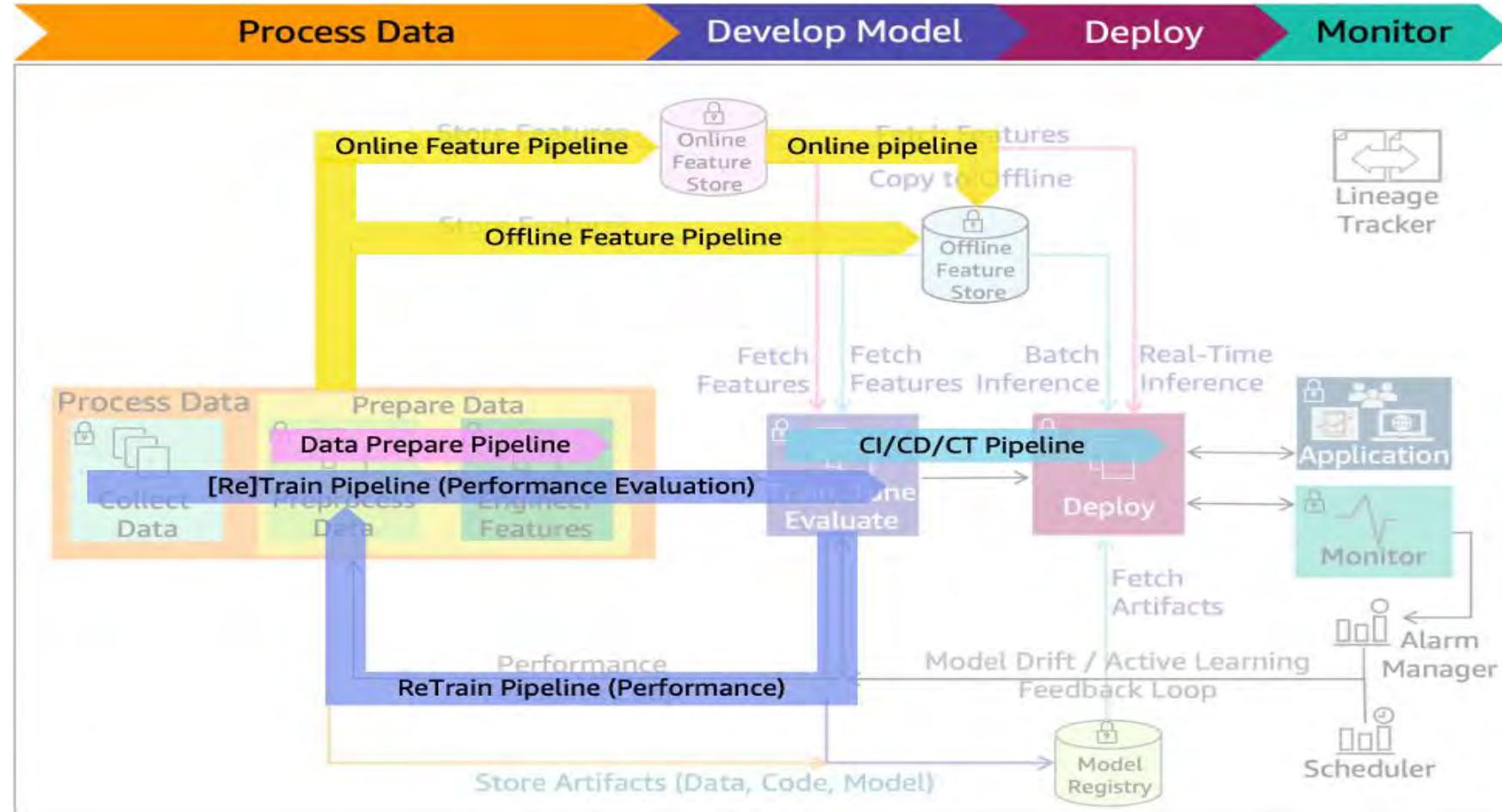
ML lifecycle phase – Model development(5)

Model evaluation

- After the model has been trained, evaluate it for its performance and accuracy
 - generate multiple models using different methods and evaluate the effectiveness of each model
 - for multiclass models, determine error rates for each class separately
- Can evaluate model using historical data (offline evaluation) or live data (online evaluation)
- Offline evaluation
 - the trained model is evaluated with a portion of the dataset that has been set aside as a holdout set
 - never used for model training or validation—it's only used to evaluate errors in the final model
 - The holdout data annotations must have high accuracy for the evaluation to make sense
 - Allocate additional resources to verify the accuracy of the holdout data
- Based on the evaluation results, might fine-tune the data, the algorithm, or both
 - may apply the concepts of data cleansing, preparation, and feature engineering again

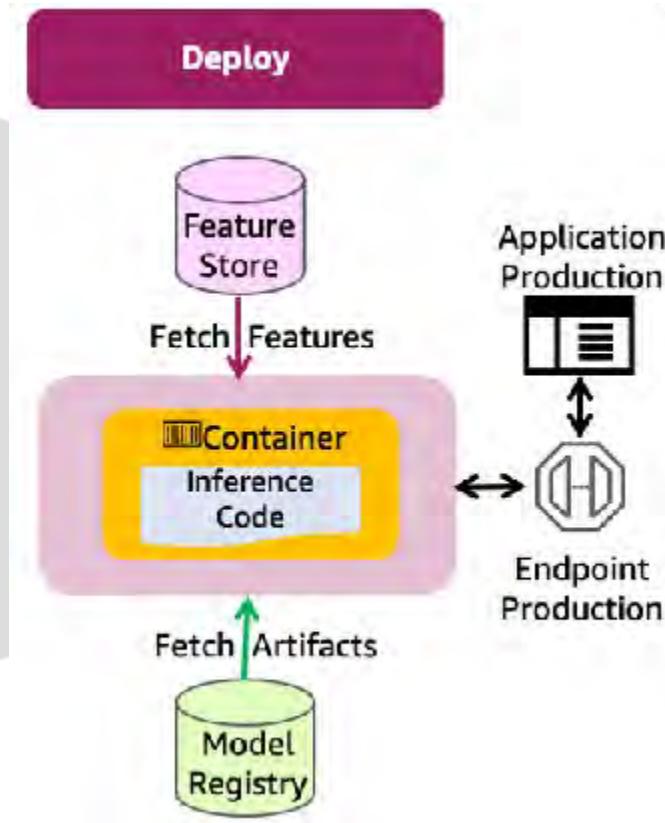
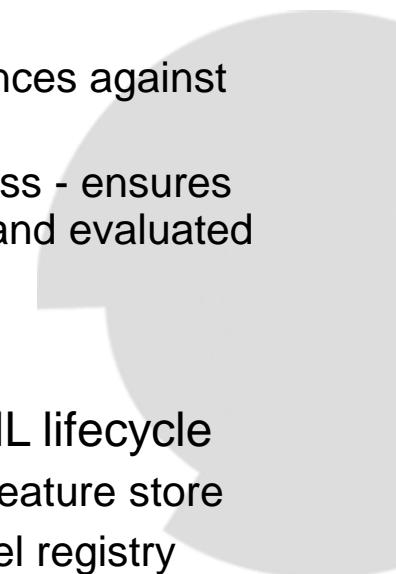
ML lifecycle phase – Model development(6)

ML lifecycle with performance evaluation pipeline added



ML lifecycle phase - Deployment

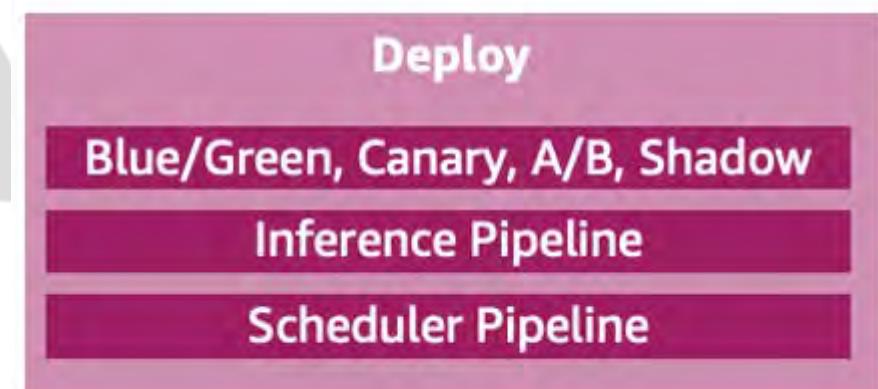
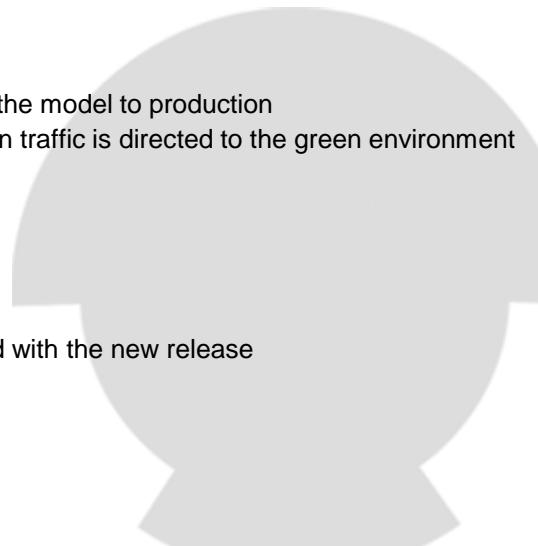
- After model have been trained, tuned, and evaluated, then can deploy the model into production
 - can make predictions and inferences against the deployed model
 - Use a manual governance process - ensures the model has fully been tested and evaluated before it's released to production
- Deploy production phase of the ML lifecycle
 - Features are retrieved from the feature store
 - Artifacts are taken from the model registry
 - Inference code used container from the container repository



ML lifecycle phase – Deployment(2)

Deployment main components

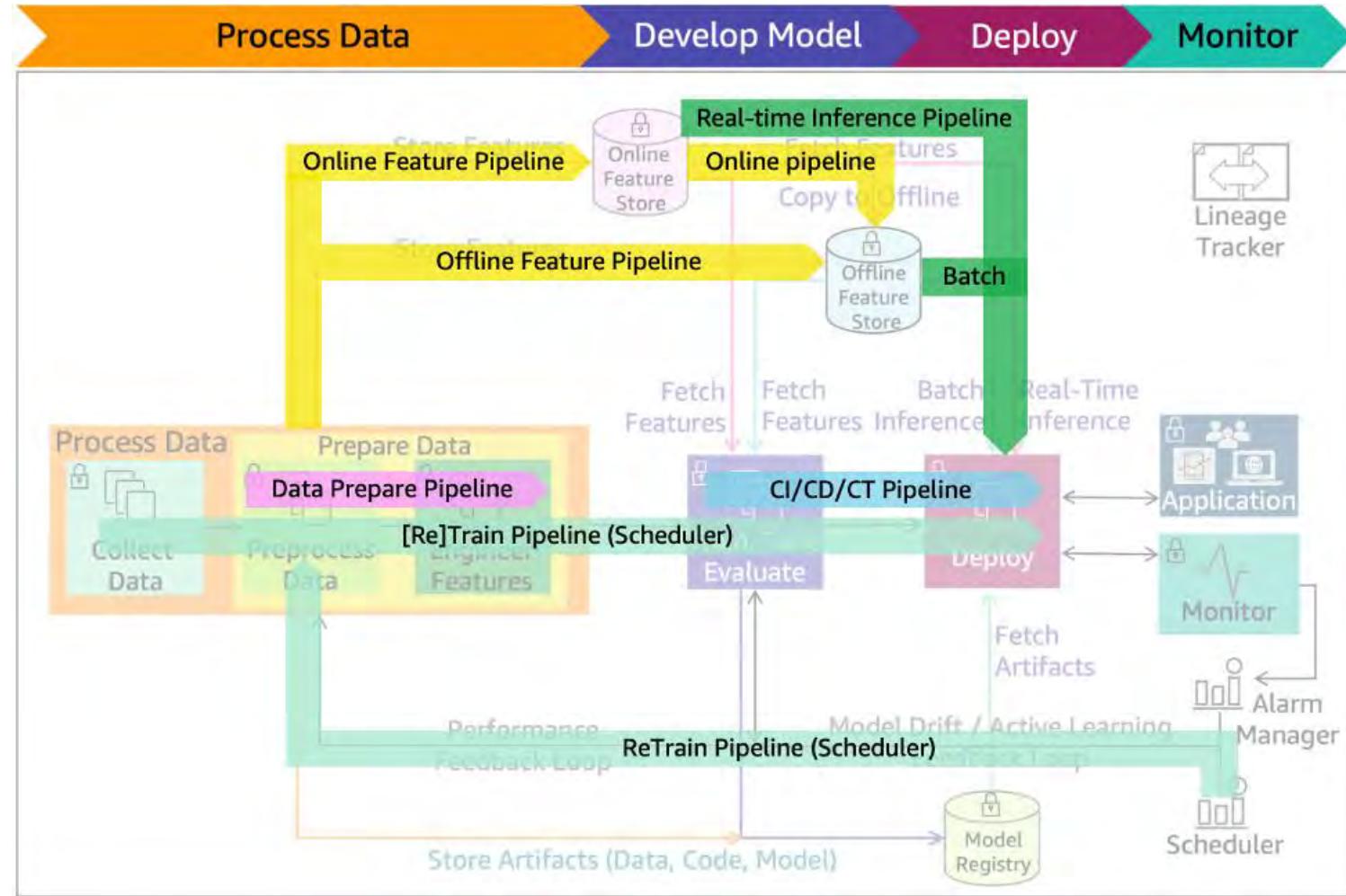
- Blue/green, canary, A/B, shadow deployment/testing
 - Deployment and testing strategies that reduce downtime and risks when releasing a new or updated version
- The blue/green deployment technique
 - provides two identical production environments
 - can use this technique when need to deploy a new version of the model to production
 - Once testing is done on the green environment, live application traffic is directed to the green environment
 - Blue environment is maintained as backup
- A canary deployment
 - first deploy the new release to a small group of users
 - Other users continue to use the previous version until satisfied with the new release
 - Then, can gradually roll the new release out to all users
- A/B testing strategy
 - enables deploying changes to a model
 - Direct a defined portion of traffic to the new model. Direct the remaining traffic to the old model
 - similar to canary testing, but has larger user groups and a longer time scale, typically days or even weeks
- Shadow deployment strategy
 - New version is available alongside the old version
 - The input data is run through both versions
 - The older version is used for servicing the production and the new one is used for testing and analysis



ML lifecycle phase – Deployment(3)

ML lifecycle with scheduler re-train, and batch/real-time Inference pipelines

- Inference pipeline
 - automates capturing of the prepared data, performing predictions and post-processing for real-time or batch inferences
- Scheduler pipeline
 - Deployed model is representative of the latest data patterns
 - Re-training at intervals can minimize the risk of data and concept drifts
 - A scheduler can initiate a re-training at business defined intervals
 - Data prepare, CI/CD/CT, and feature pipelines will also be active during this process



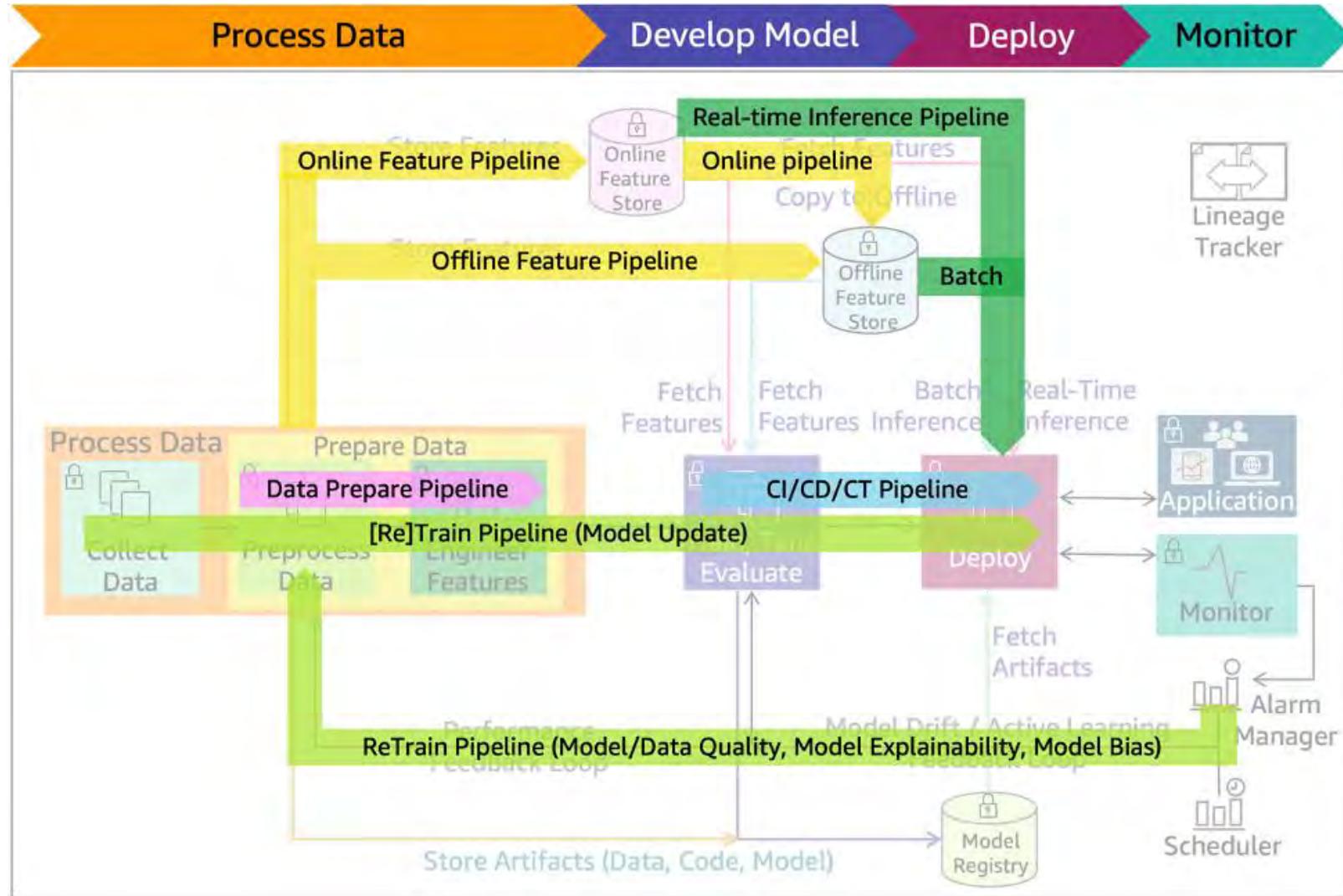
ML lifecycle phase – Monitoring

Post deployment monitor main components

- The model monitoring system must
 - capture data,
 - compare that data to the training set, define rules to detect issues
 - and send alerts
- Process repeats on a defined schedule, when initiated by an event, or when initiated by human intervention
 - The issues detected in the monitoring phase include: data quality, model quality, bias drift, and feature attribution drift
- Key components of monitoring including:
 - Model explainability
 - uses explainability to evaluate the soundness of the model and if the predictions can be trusted
 - Detect drift
 - detects data and concept drifts
 - Data drift is the significant changes to data distribution compared to data used for training
 - Concept drift is when the properties of target variables change
 - Any kind of drift will result in model performance degradation
 - will initiate an alert and send it to alarm manager system
 - Model update pipeline
 - If alarm manager identifies any violations, it launches the model update pipeline for a re-train
 - Data prepare, CI/CD/CT, and feature pipelines will also be active during this process

ML lifecycle phase – Monitoring(2)

ML lifecycle with model update re-train and batch/real-time Inference pipelines

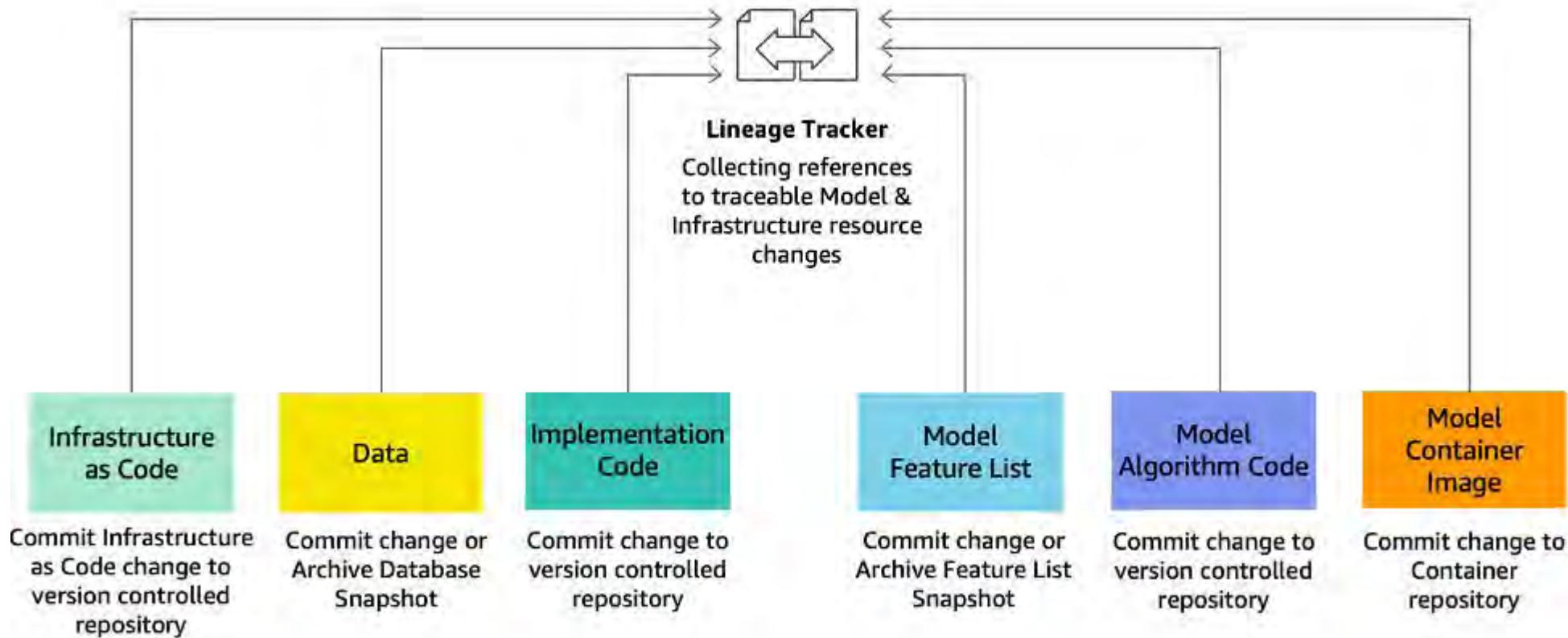


Additional - Lineage tracker

Lineage tracker

- Requirement
 - enables reproducible machine learning experiences
 - enables re-creating the ML environment at a specific point-in-time, reflecting the versions of all resources and environments at that time
 - collects references to traceable data, model and infrastructure resource changes
- Components:
 - System architecture (infrastructure as code to address environment drift)
 - Data (metadata, values, and features)
 - Model (algorithm, features, parameters, and hyperparameters)
 - Code (implementation, modeling, and pipeline)
- Working:
 - Collects changed references through alternative iterations of ML lifecycle phases
 - Alternative algorithms and feature lists are evaluated as experiments for final production deployment
 - The collected information enables going back to a specific point-in-time release and recreate it

Lineage tracker



Lineage tracker

Components

- Infrastructure as code (IaC)
 - Modeling, provisioning, and managing cloud computing resources (compute, storage, network, and application services) can be automated using IaC
 - eliminates configuration drift through automation, while increasing the speed and agility of infrastructure deployments
 - IaC code changes are committed to version-controlled repository
- Data
 - Store data schemes and metadata in version control systems
 - Store the data in a storage media like a data lake
 - The location or link to the data can be in a configuration file and stored in code version control media
- Implementation code
 - Changes to any implementation code at any point-in-time can be stored in version control media
- Model feature list
 - Feature store technology referenced in the scenario architecture diagrams stores features as well as their versions for any point-in-time changes
- Model algorithm code
 - Changes to any model algorithm code at any point-in-time can be stored in version control media
- Model container image
 - Versions of model container images for any point-in-time changes can be stored in container repositories managed by container registry



Thank You!

In our next session:



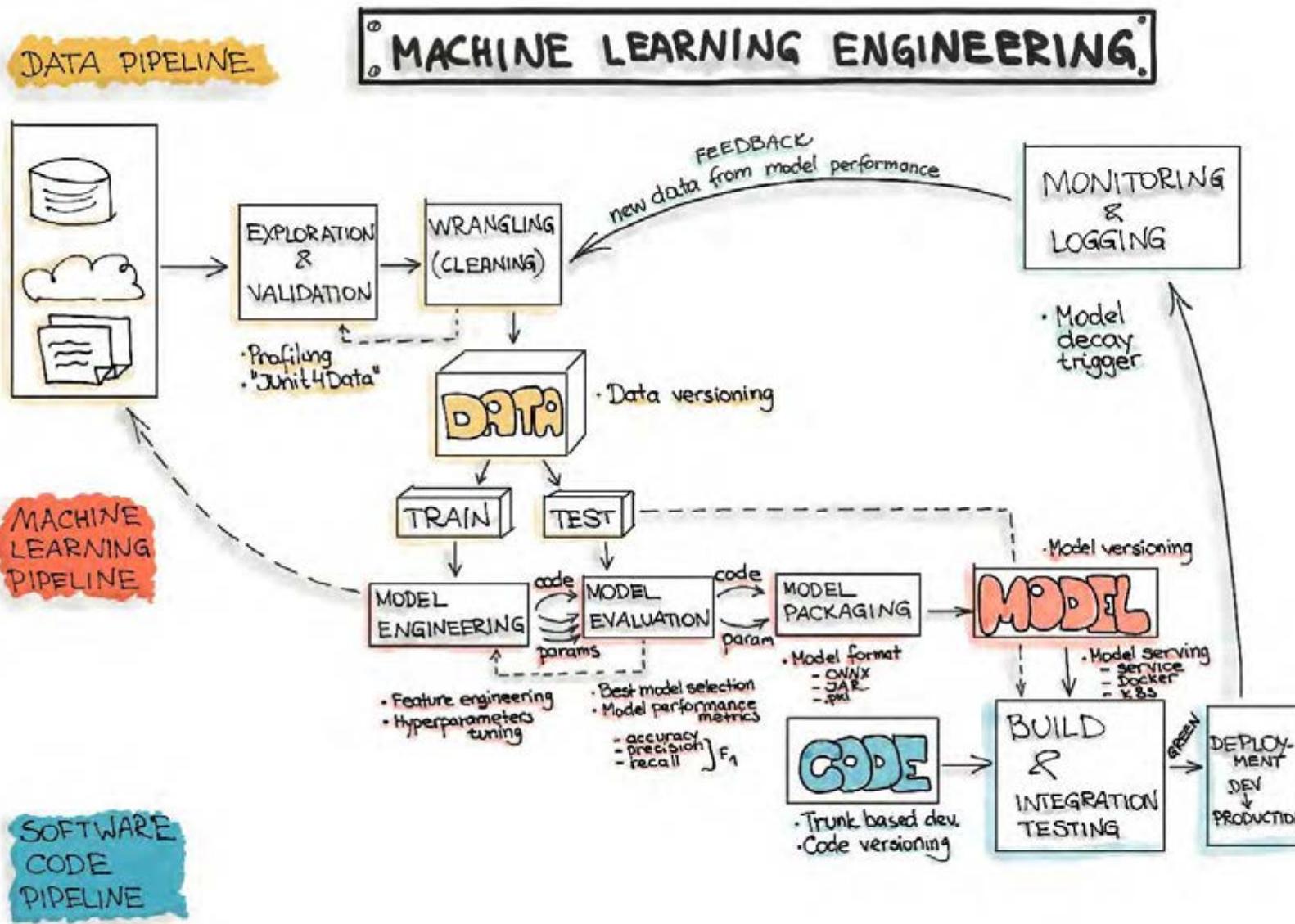
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Data: Data Engineering Pipelines

Pravin Y Pawar

Extracted from
[Three Levels of ML Software](#)

Machine Learning Workflow



Data: Data Engineering Pipelines

- Fundamental part of any machine learning workflow is Data
 - Collecting good data sets has a huge impact on the quality and performance of the ML model
- “Garbage In, Garbage Out”,
 - In the ML context means that the ML model is only as good as data using which its built
 - Data indirectly influence the overall performance of the production system
 - The amount and quality of the data set are usually problem-specific and can be empirically discovered
- Data engineering is reported as heavily time-consuming
 - might spend the majority of time on a machine learning project constructing data sets, cleaning, and transforming data
 - includes a sequence of operations on the available data
 - final goal of these operations is to create training and testing datasets for the ML algorithms
- Stages of the data engineering pipeline
 - Data Ingestion
 - Exploration and Validation
 - Data Wrangling (Cleaning)
 - Data Splitting

Data Ingestion

- Collecting data by using various systems, frameworks and formats
 - such as internal/external databases, data marts, OLAP cubes, data warehouses, OLTP systems, Spark, HDFS etc
 - might also include synthetic data generation or data enrichment
- Contains actions that should be maximally automated:
 - Data Sources Identification: Find the data and document its origin (data provenance)
 - Space Estimation: Check how much storage space it will take
 - Space Location: Create a workspace with enough storage space
 - Obtaining Data: Get the data and convert them to a format that can be easily manipulated without changing the data itself
 - Back up Data: Always work on a copy of the data and keep the original dataset untouched
 - Privacy Compliance: Ensure sensitive information is deleted or protected (e.g., anonymized) to ensure GDPR compliance
 - Metadata Catalog: Start documenting the metadata of the dataset by recording the basic information about the size, format, aliases, last modified time, and access control lists
 - Test Data: Sample a test set, put it aside, and never look at it to avoid the “data snooping” bias

Exploration and Validation

- Exploration includes data profiling to obtain information about the content and structure of the data
 - The output of this step is a set of metadata, such as max, min, avg of values
- Data validation operations are user-defined error detection functions, which scan the dataset to spot some errors
 - process of assessing the quality of the data by running dataset validation routines (error detection methods)
- Includes actions:
 - Use RAD tools: Using Jupyter notebooks is a good way to keep records of data exploration and experimentation
 - Attribute Profiling: Obtain and document the metadata about each attribute, such as
 - Name
 - Number of Records
 - Data Type (categorical, numerical, int/float, text, structured, etc.)
 - Numerical Measures (min, max, avg, median, etc. for numerical data)
 - Amount of missing values (or “missing value ratio” = Number of absent values/ Number of records)
 - Type of distribution (Gaussian, uniform, logarithmic, etc.)
 - Label Attribute Identification: For supervised learning tasks, identify the target attribute(s).
 - Data Visualization: Build a visual representation for value distribution
 - Attributes Correlation: Compute and analyze the correlations between attributes
 - Additional Data: Identify data that would be useful for building the model (go back to “Data Ingestion”)

Data Wrangling (Cleaning)

- Is a Data preparation step where we programmatically wrangle data,
 - e.g., by re-formatting or re-structuring particular attributes that might change the form of the data's schema
 - recommended to write scripts or functions for all data transformations in the data pipeline to reuse all these functionalities on future data
- Includes:
 - Transformations: Identify the promising transformations you may want to apply
 - Outliers: Fix or remove outliers (optional)
 - Missing Values: Fill in missing values (e.g., with zero, mean, median) or drop their rows or columns
 - Not relevant Data: Drop the attributes that provide no useful information for the task (relevant for feature engineering)
 - Restructure Data: Might include the following operations (from the book "Principles of Data Wrangling")
 - Reordering record fields by moving columns
 - Creating new record fields through extracting values
 - Combining multiple record fields into a single record field
 - Filtering datasets by removing sets of records
 - Shifting the granularity of the dataset and the fields associated with records through aggregations and pivots

Data Splitting

- Splitting the data into
 - Training
 - Validation
 - Test
- datasets to be used during the core machine learning stages to produce the ML model



Thank You!

In our next session:



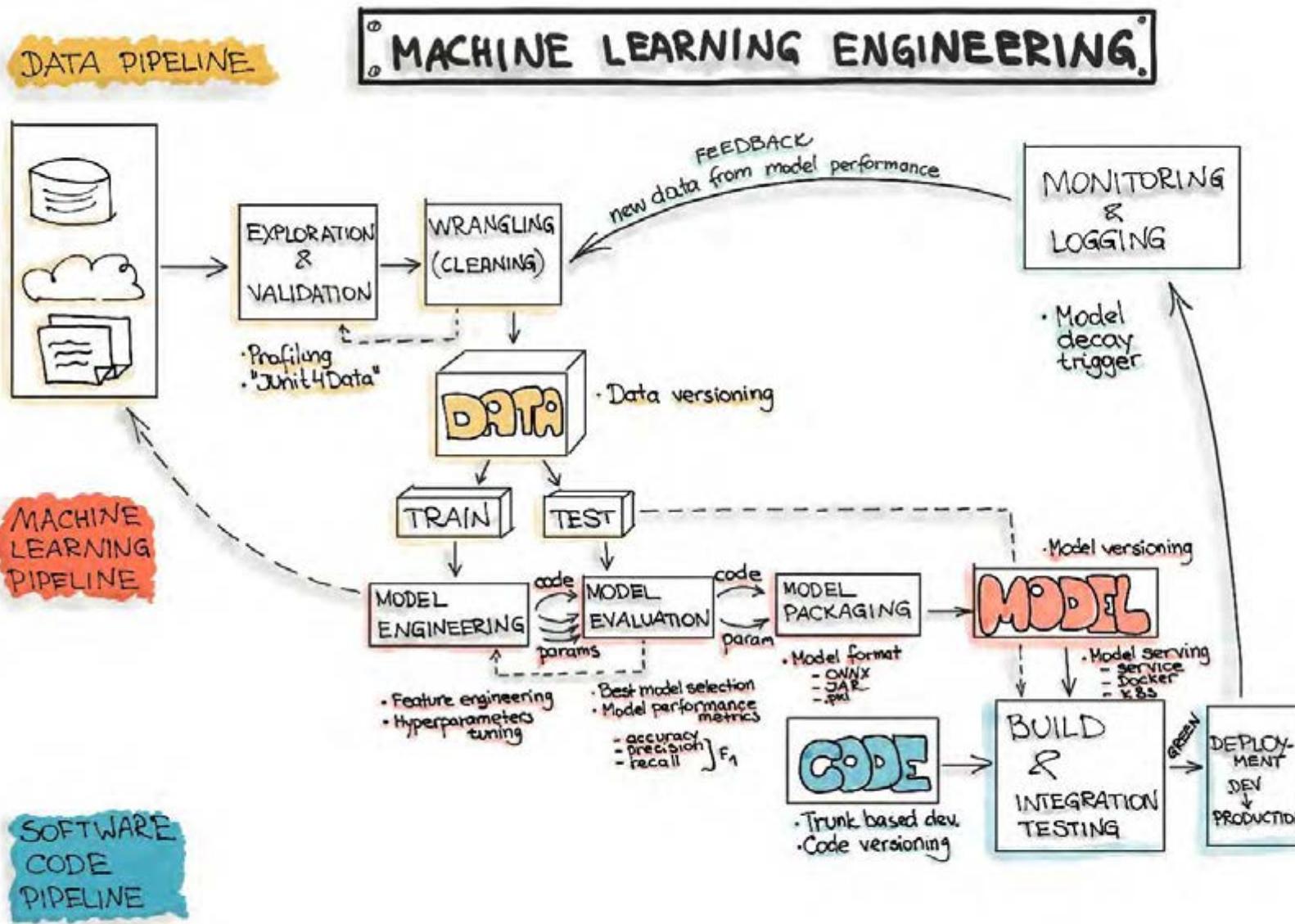
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Model: Machine Learning Pipelines

Pravin Y Pawar

Extracted from
[Three Levels of ML Software](#)

Machine Learning Workflow



Model: Machine Learning Pipelines

- The core of the ML workflow is the phase of writing and executing machine learning algorithms to obtain an ML model
- The model engineering pipeline is
 - usually utilized by a data science team
 - includes a number of operations that lead to a final model
- Operations include
 - Model Training
 - Model Evaluation
 - Model Testing
 - Model Packaging
- Recommended to automate these steps as much as possible!

Model Training

Feature engineering

- The process of applying the machine learning algorithm on training data to train an ML model
 - includes feature engineering the hyperparameter tuning for the model training activity
- Feature engineering might include:
 - Discretize continuous features
 - Decompose features (e.g., categorical, date/time, etc.)
 - Add transformations of features (e.g., $\log(x)$, \sqrt{x} , x^2 , etc.)
 - Aggregate features into promising new features
 - Feature scaling: Standardize or normalize features
 - New features should be added quickly to get fast from a feature idea to the feature running in production

Model Training(2)

Model Engineering

- Might be an iterative process and include the following workflow:
 - Every ML model specification (code that creates an ML model) should go through a code review and be versioned.
 - Train many ML models from different categories
 - (e.g., linear regression, logistic regression, k-means, naive Bayes, SVM, Random Forest, etc.) using standard parameters
 - Measure and compare their performance
 - For each model, use N-fold cross-validation and compute the mean and standard deviation of the performance measure on the N folds.
 - Error Analysis
 - analyze the types of errors the ML models make.
 - Consider further feature selection and engineering
 - Identify the top three to five most promising models, preferring models that make different types of errors.
 - Hyperparameters tuning by using cross-validation.
 - Consider Ensemble methods such as majority vote, bagging, boosting, or stacking
 - Combining ML models should produce better performance than running them individually

Model Evaluation, Testing and Packaging

- Model Evaluation
 - Validate the trained model to ensure it meets original business objectives before serving the ML model in production to the end-user
- Model Testing
 - Once the final ML model is trained, its performance needs to be measured
 - by using the hold-back test dataset to estimate the generalization error by performing the final “Model Acceptance Test”
- Model Packaging
 - The process of exporting the final ML model into a specific format (e.g. PMML, PFA, or ONNX)
 - which describes the model to be consumed by the business application

ML Model serialization formats

- Various formats to distribute ML models
 - In order to achieve a distributable format, the ML model should be present and should be executable as an independent asset
 - means that the ML models should work outside of the model-training environment
- Two flavors
 - Language-agnostic exchange formats
 - Vendor-specific exchange formats
- Language-agnostic exchange formats
 - Amalgamation is the simplest way to export an ML model as portable - model and all necessary code to run are bundled as one package
 - Usually, a single source code file that can be compiled on nearly any platform as a standalone program
 - Create a standalone version of an ML model by using SKompiler to transform trained Scikit-learn models into other forms,
 - such as SQL queries, Excel formulas,
 - Portable Format for Analytics (PFA) files
 - or SymPy expressions - translated to code in a variety of languages, such as C, Javascript, Rust, Julia, etc.

ML Model serialization formats(2)

Common Language-agnostic exchange formats

- PMML
 - format for model serving based on XML with the file extension .pmml - standardized by the Data Mining Group (DMG)
 - Basically, .pmml describes a model and pipeline in XML
 - The PMML supports not all of the ML algorithms, and its usage in open source-driven tools is limited due to licensing issues
- PFA (Portable Format for Analytics) - designed as a replacement for PMML
 - is a string of JSON-formatted text that describes an executable called a scoring engine
 - each engine has a well-defined input, a well-defined output, and functions for combining inputs to construct the output in an expression-centric syntax tree
 - To run ML models as PFA files, need a PFA-enabled environment
- ONNX (Open Neural Network eXchange)
 - ML framework independent file format
 - created to allow any ML tool to share a single model format - supported by many big tech companies such as Microsoft, Facebook, and Amazon
 - Once the ML model is serialized in the ONNX format,
 - can be consumed by onnx-enabled runtime libraries (also called inference engines) and then make predictions

ML Model serialization formats(3)

Vendor-specific exchange formats

- Scikit-Learn
 - saves models as pickled python objects, with a .pkl file extension
- H2O
 - allows to convert the models to either POJO (Plain Old Java Object) or MOJO (Model Object, Optimized)
- SparkML models
 - can be saved in the MLeap file format and served in real-time using an MLeap model server
 - The MLeap runtime is a JAR that can run in any Java application
 - supports Spark, Scikit-learn, and Tensorflow for training pipelines and exporting them to an MLeap Bundle
- TensorFlow
 - saves models as .pb files; which is the protocol buffer files extension
- PyTorch
 - serves models by using their proprietary Torch Script as a .pt file
 - can be served from a C– application
- Keras
 - saves a model as a .h5 file, which is known as a data file saved in the Hierarchical Data Format (HDF)- contains multidimensional arrays of data
- Apple
 - has its proprietary file format with the extension .mlmodel to store models embedded in iOS applications
 - The Core ML framework has native support for Objective-C and Swift programming languages
 - Applications trained in other ML frameworks, such as TensorFlow, Scikit-Learn, and other frameworks need to use tools like coremltools and Tensorflow converter to translate their ML model files to the .mlmodel format for use on iOS



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Different forms of ML workflows

Pravin Y Pawar

Extracted from
[Three Levels of ML Software](#)

Different forms of ML workflows

- Operating an ML model might assume **several architectural styles**
- Primarily **Four architectural patterns** which are classified along two dimensions:
 - ML Model Training
 - ML Model Prediction
- For sake of simplicity disregard the third dimension - **ML Model Type**
 - which denotes the type of machine learning algorithm such as
 - Supervised
 - Unsupervised
 - Semi-supervised
 - and Reinforcement Learning

Model Training Patterns

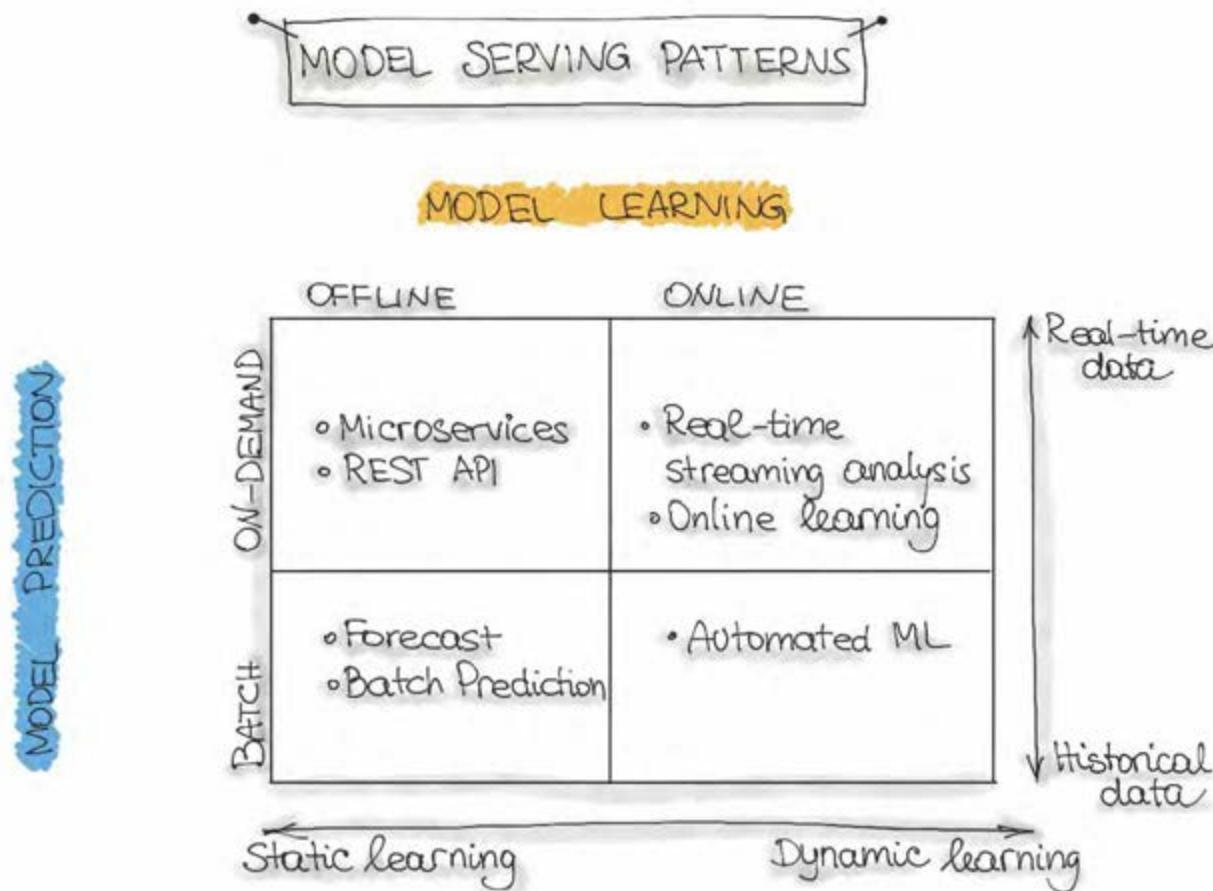
- Two ways to perform ML Model Training:
 - Offline learning
 - Online learning
- Offline learning (aka batch or static learning)
 - The model is trained on a set of already collected data
 - After deploying to the production environment, the ML model remains constant until it re-trained
 - Model will see a lot of real-live data and becomes stale - 'model decay' and should be carefully monitored
- Online learning (aka dynamic learning)
 - The model is regularly being re-trained as new data arrives, e.g. as data streams
 - Usually the case for ML systems that use time-series data, such as sensor, or stock trading data to accommodate the temporal effects in the ML model

Model Prediction Patterns

- Two modes to denote the mechanics of the ML model to makes predictions
 - Batch predictions
 - Real-time predictions
- Batch predictions
 - The deployed ML model makes a set of predictions based on historical input data
 - often sufficient for data that is not time-dependent, or when it is not critical to obtain real-time predictions as output
- Real-time predictions (aka on-demand predictions)
 - Predictions are generated in real-time using the input data that is available at the time of the request

ML architecture patterns

Forecast, Web-Service, Online Learning, and AutoML



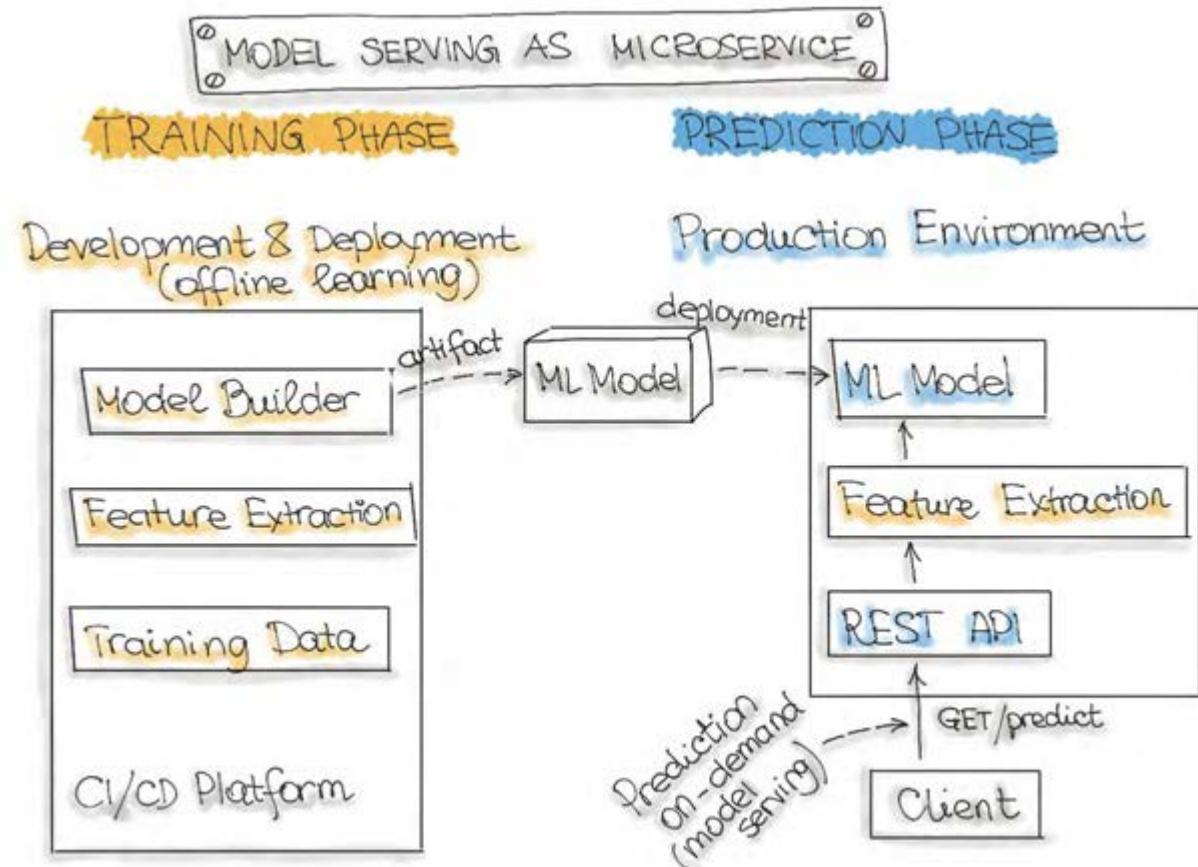
Forecast

- Widely spread in academic research or data science education (e.g., Kaggle or DataCamp)
 - used to experiment with ML algorithms and data as it is the easiest way to create a machine learning system
 - not very useful in an industry setting for production systems (e.g. mobile applications)
- Usually involves steps
 - take an available dataset
 - train the ML model
 - run this model on another (mostly historical) data
 - makes predictions

Web-Service (or Microservices)

Architecture for wrapping trained models as deployable services

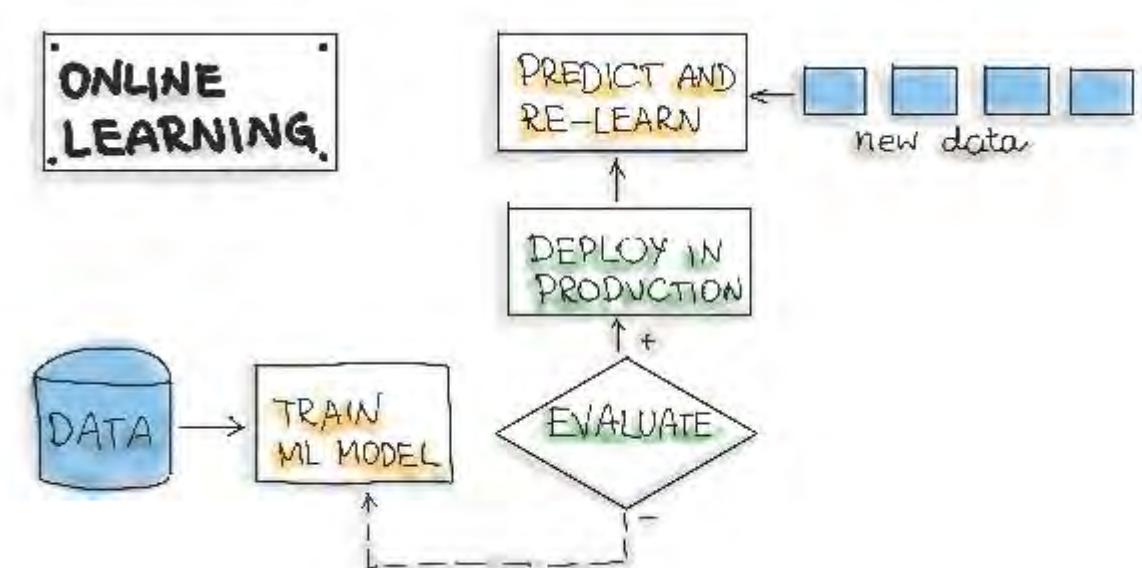
- The most commonly described deployment architecture for ML models
- The web service takes input data and outputs a prediction for the input data points
 - Model is trained offline on historical data, but it uses real-live data to make predictions
 - Model remains constant until it is re-trained and re-deployed into the production system.
- The difference from a forecast (batch predictions) is that
 - the ML model runs near real-time
 - handles a single record at a time instead of processing all the data at once



Online Learning (Real-time streaming analytics)

Most dynamic way to embed machine learning into a production system

- Confusing name because the core learning or ML model training is usually not performed on the live system
 - call it incremental learning- term online learning is already established within the ML community
- In this type of ML workflow
 - ML learning algorithm is continuously receiving a data stream, either as single data points or in small groups called mini-batches
 - System learns about new data on the fly as it arrives, so the ML model is incrementally being re-trained with new data
 - Continually re-trained model is instantly available as a web service
 - Technically works well with the lambda architecture in big data systems
 - Model would typically run as a service on a Kubernetes cluster or similar
- A big difficulty with the online learning system in production is that
 - if bad data is entering the system, the ML model, as well as the whole system performance, will increasingly decline



AutoML

Sophisticated version of online learning

- AutoML is getting a lot of attention
 - considered the next advance for enterprise ML
 - promises training ML models with minimal effort and without machine learning expertise
 - User needs to provide data, and the AutoML system automatically selects an ML algorithm and configures the selected algorithm
 - very experimental way to implement ML workflows
 - usually provided by big cloud providers, such as Google or MS Azure
 - Instead of updating the model, need to execute an entire ML model training pipeline in production that results in new models on the fly



Thank You!

In our next session:



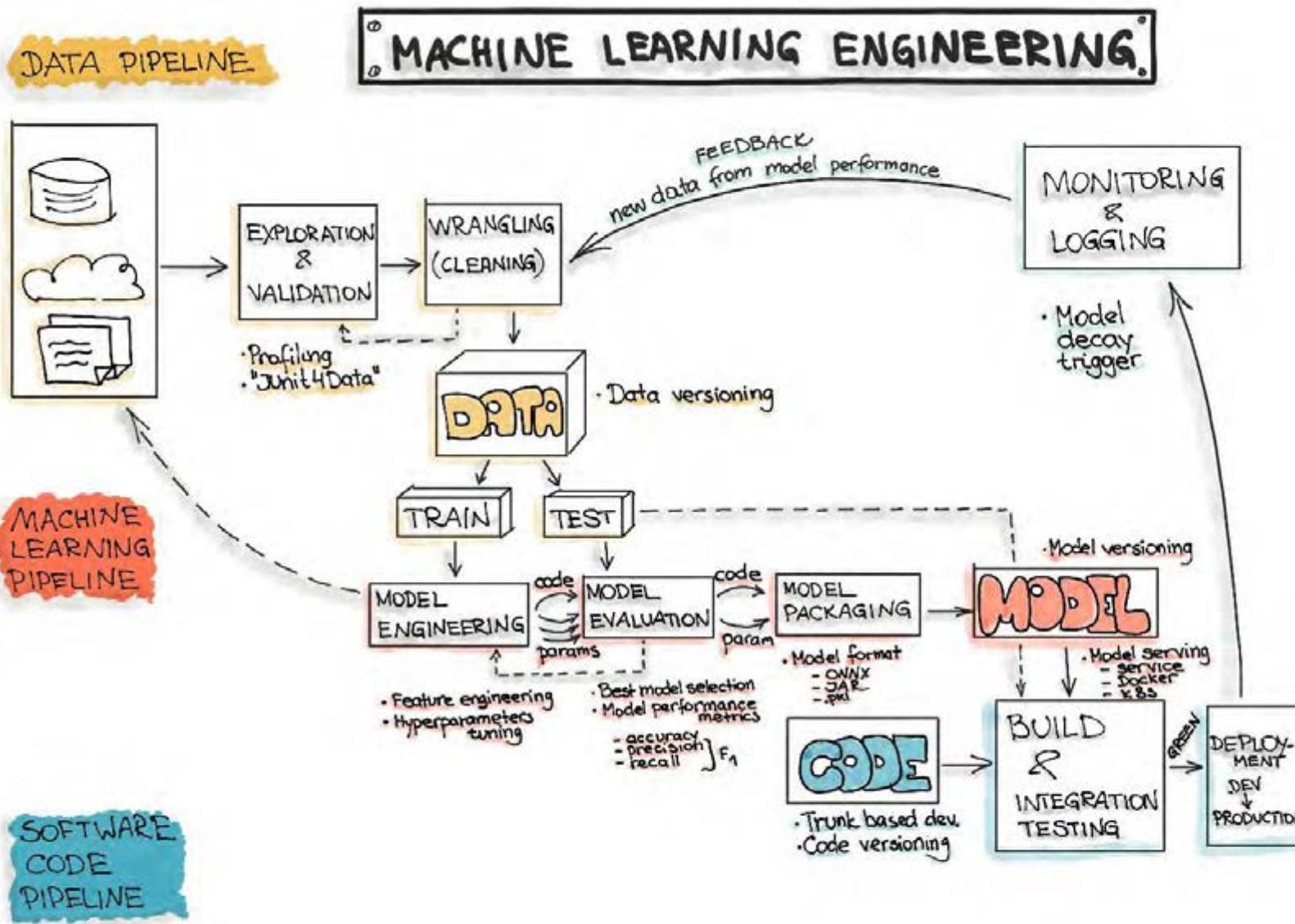
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Code: Deployment Pipelines

Pravin Y Pawar

Extracted from
[Three Levels of ML Software](#)

Machine Learning Workflow



Code: Deployment Pipelines

- The final stage of delivering an ML project includes the following three steps:
 - Model Serving
 - Model Performance Monitoring
 - Model Performance Logging
- Model Serving
 - The process of deploying the ML model in a production environment
- Model Performance Monitoring
 - The process of observing the ML model performance based on live and previously unseen data, such as prediction or recommendation
 - interested in ML-specific signals, such as prediction deviation from previous model performance
 - signals might be used as triggers for model re-training
- Model Performance Logging
 - Every inference request results in a log-record.

Model Serving Patterns

- Three components should be considered when ML model is served in a production environment
 - The inference is the process of getting data to be ingested by a model to compute predictions
 - requires a model, an interpreter for the execution, and input data
- Deploying an ML system to a production environment includes two aspects,
 - First deploying the pipeline for automated retraining and ML model deployment
 - Second, providing the API for prediction on unseen data
- Model serving is a way to integrate the ML model in a software system
- Five patterns to put the ML model in production:
 - Model-as-Service
 - Model-as-Dependency
 - Precompute
 - Model-on-Demand
 - and Hybrid-Serving

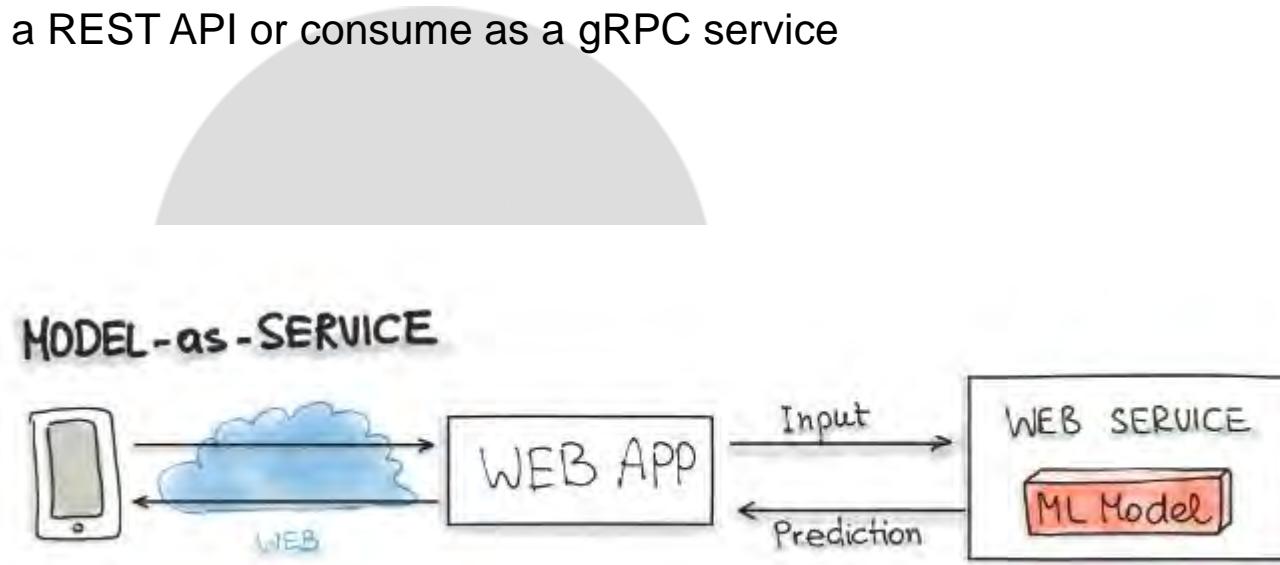
Model Serving Patterns(2)

Machine Learning Model Serving Taxonomy

Machine Learning Model Serving Taxonomy			
	ML Model		
Service & Versioning	Together with the consuming application	Independent from the consuming application	
Compile/ Runtime Availability	Build & runtime available	Available remotely through REST API/RPC	Available at the runtime scope
Serving Patterns	"Model-as-Dependency"	"Model-as-Service"	"Precompute" and "Model on Demand"
	Hybrid Model Serving (Federated Learning)		

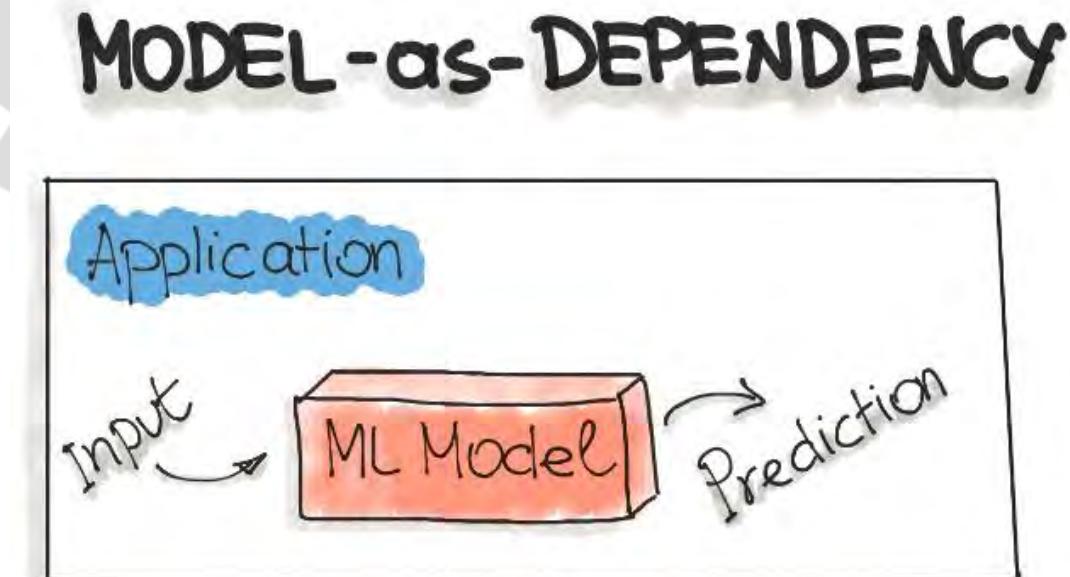
Model-as-Service

- A common pattern for wrapping an ML model as an independent service
 - can wrap the ML model and the interpreter within a dedicated web service
 - applications can request through a REST API or consume as a gRPC service
- Used for various ML workflows
 - Forecast
 - Web Service
 - Online Learning



Model-as-Dependency

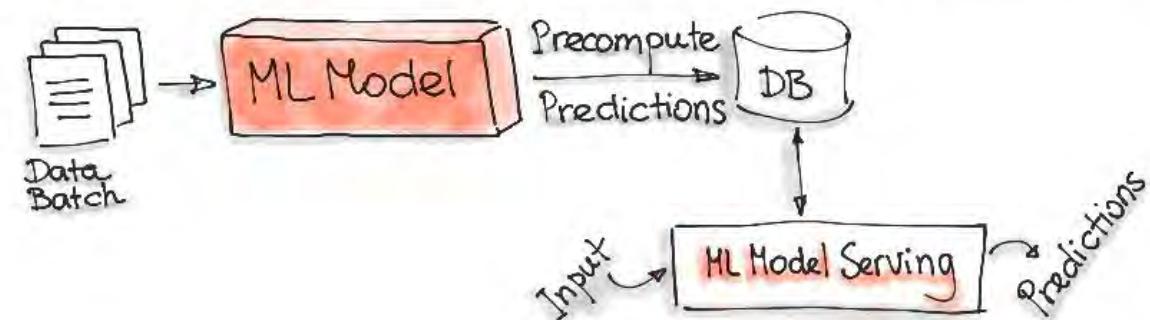
- Probably the most straightforward way to package an ML model
 - mostly used for implementing the Forecast pattern.
- A packaged ML model is considered as a dependency within the software application
 - For example, the application consumes the ML model like a conventional jar dependency by invoking the prediction method and passing the values
 - The return value of such method execution is some prediction that is performed by the previously trained ML model



Precompute Serving Pattern

- Tightly related to the Forecast ML workflow
- Use an already trained ML model and precompute the predictions for the incoming batch of data
 - Resulting predictions are persisted in the database
 - For any input request, query the database to get the prediction result

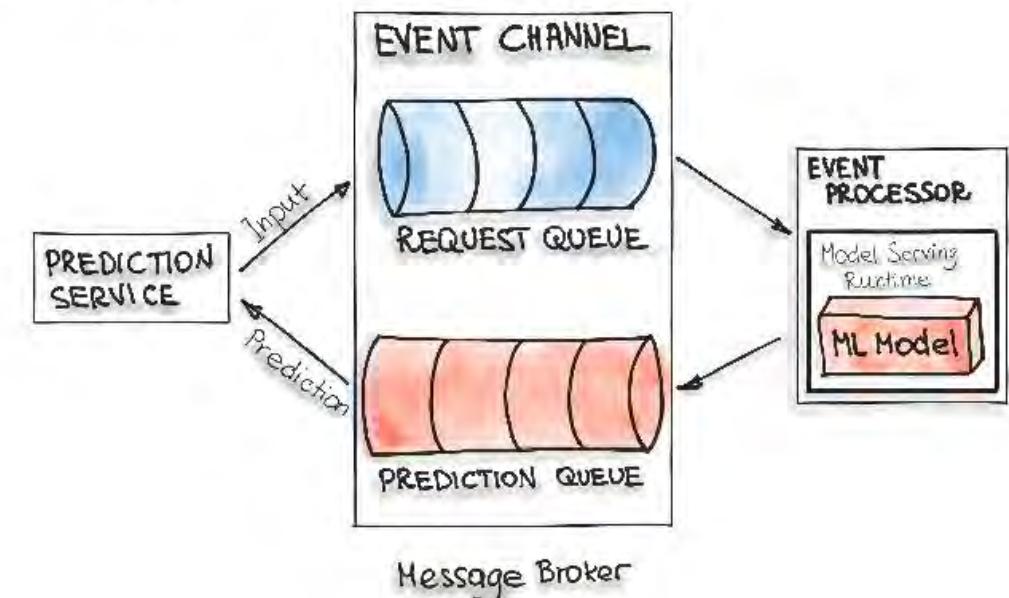
PRECOMPUTE SERVING PATTERN



Model-on-Demand

- Treats the ML model as a dependency that is available at runtime
 - contrary to the Model-as-Dependency pattern, has its own release cycle and is published independently
- Message-broker architecture is typically used for such on-demand model serving
 - contains two main types of architecture components:
 - a broker component
 - an event processor component
 - Broker component is the central part that contains the event channel that are utilised within the event flow
 - The event channels, which are enclosed in the broker component, are message queues
 - Message broker allows one process to write prediction-requests in a input queue
 - Event processor contains the model serving runtime and the ML model
 - connects to the broker, reads these requests in batch from the queue and sends them to the model to make the predictions
- The model serving process runs the prediction generation on the input data
 - writes the resulted predictions to the output queue
 - queued prediction results are pushed to the prediction service that initiated the prediction request

MODEL-ON-DEMAND

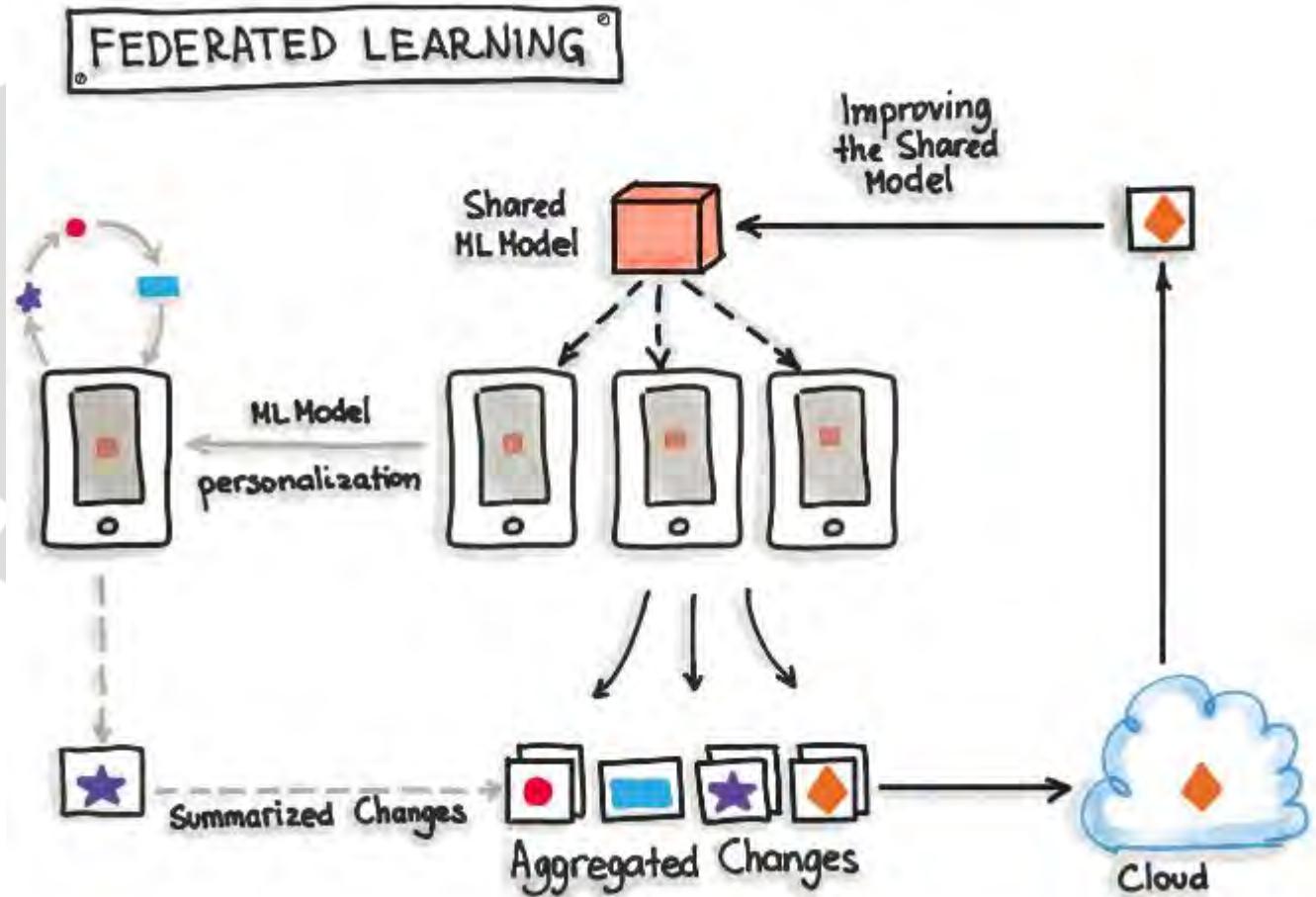


Hybrid-Serving (Federated Learning)

- Unique in the way it does, there is not only one model that predicts the outcome, but there are also lots of it
 - Exactly spoken there are as many models as users exist, in addition to the one that's held on a server
- Start with the unique model, the one on the server
 - Model on the server-side is trained only once with the real-world data
 - sets the initial model for each user
 - a relatively general trained model so it fits for the majority of users
- On the other side, there are the user-side models - really unique models
 - possible for the devices to train their own models due to hardware enhancements
 - devices will train their own highly specialized model for their own user
 - Once in a while, the devices send their already trained model data (not the personal data) to the server
- Then the server model will be adjusted
 - the actual trends of the whole user community will be covered by the model
 - this updated server model is set to be the new initial model that all devices are using

Hybrid-Serving (Federated Learning) 2

- Not having any downsides for the users, while the server model gets updated, this happens only when the device is idle, connected to WiFi and charging
 - testing is done on the devices, therefore the newly adopted model from the server is sent to the devices and tested for functionality
- Big benefit
 - data used for training and testing, which is highly personal, never leaves the devices while still capturing all data that is available
 - possible to train highly accurate models while not having to store tons of (probably personal) data in the cloud
- Constraints
 - mobile devices are less powerful
 - the training data is distributed across millions of devices and these are not always available for training
 - Exactly for this TensorFlow Federated (TFF) has been created - lightweight form of TensorFlow

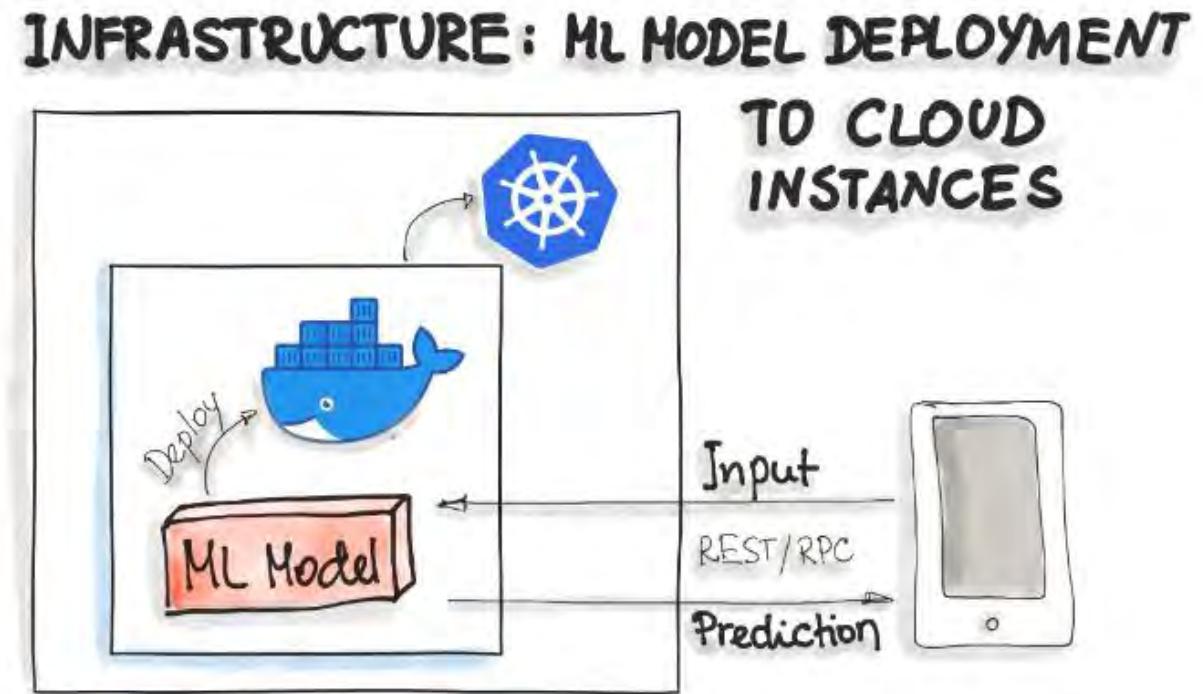


Deployment Strategies

- Common ways for wrapping trained models as deployable services, namely deploying ML models as
 - Docker Containers to Cloud Instances
 - Serverless Functions

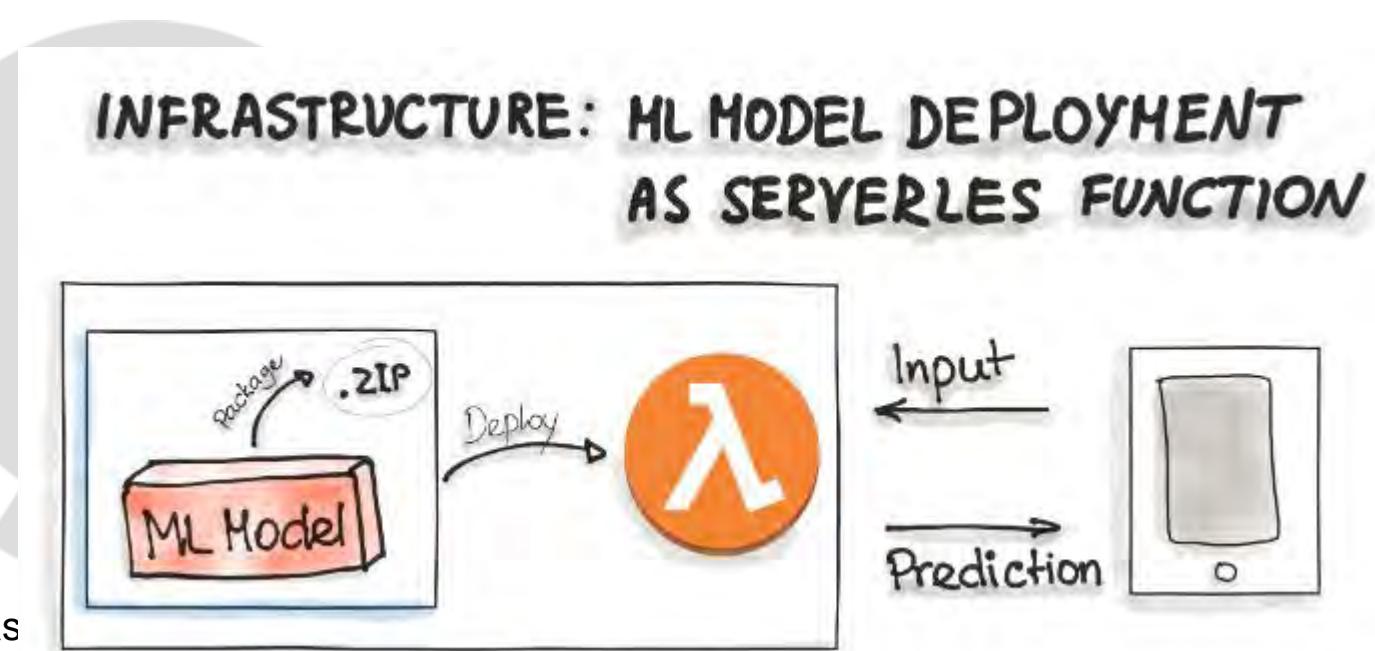
Deploying ML Models as Docker Containers

- No standard, open solution to ML model deployment!
- Containerization becomes the de-facto standard for delivery
 - as ML model inference being considered stateless, lightweight, and idempotent
 - means deploy a container that wraps an ML model inference code
- Docker is considered to be de-facto standard containerization technology
 - for on-premise, cloud, or hybrid deployments
- One ubiquitous way is to package the whole ML tech stack (dependencies) and the code for ML model prediction into a Docker container
 - Then Kubernetes or an alternative (e.g. AWS Fargate) does the orchestration
 - The ML model functionality, such as prediction, is then available through a REST API (e.g. implemented as Flask application)



Deploying ML Models as Serverless Functions

- Various cloud vendors already provide machine-learning platforms - can deploy model with their services
 - Amazon AWS Sagemaker, Google Cloud AI Platform, Azure Machine Learning Studio, and IBM Watson Machine Learning
 - Commercial cloud services also provide containerization of ML models such as AWS Lambda and Google App Engine servlet host
- In order to deploy an ML model as a serverless function,
 - the application code and dependencies are packaged into .zip files, with a single entry point function
 - could be managed by major cloud providers such as Azure Functions, AWS Lambda, or Google Cloud Functions
 - However, attention should be paid to possible constraints of the deployed artifacts such as the size of the artifacts





Thank You!

In our next session:

Challenges in Deploying Machine Learning: a Survey of Case Studies

Andrei Paleyes
 Department of Computer Science
 University of Cambridge
 ap2169@cam.ac.uk

Raoul-Gabriel Urma
 Cambridge Spark
 raoul@cambridgespark.com

Neil D. Lawrence
 Department of Computer Science
 University of Cambridge
 ndl21@cam.ac.uk

Abstract

In recent years, machine learning has received increased interest both as an academic research field and as a solution for real-world business problems. However, the deployment of machine learning models in production systems can present a number of issues and concerns. This survey reviews published reports of deploying machine learning solutions in a variety of use cases, industries and applications and extracts practical considerations corresponding to stages of the machine learning deployment workflow. Our survey shows that practitioners face challenges at each stage of the deployment. The goal of this paper is to layout a research agenda to explore approaches addressing these challenges.

1 Introduction

Machine learning (ML) has evolved from purely an area of academic research to an applied field. In fact, according to a recent global survey conducted by McKinsey, machine learning is increasingly adopted in standard business processes with nearly 25 percent year-over-year growth [1] and with growing interest from the general public, business leaders [2] and governments [3].

This shift comes with challenges. Just as with any other field, there are significant differences between what works in academic setting and what is required by a real world system. Certain bottlenecks and invalidated assumptions should always be expected in the course of that process. As more solutions are developed and deployed, practitioners sometimes report their experience in various forms, including publications and blog posts. In this study, we undertake a survey of these reports to capture the current challenges in deploying machine learning in production¹. First, we provide an overview of the machine learning deployment workflow. Second, we review use case studies to extract problems and concerns practitioners have at each particular deployment stage. Third, we discuss cross-cutting aspects that affect every stage of the deployment workflow: ethical considerations, end users' trust and security. Finally, we conclude with a brief discussion of potential solutions to these issues and further work.

A decade ago such surveys were already conducted, albeit with a different purpose in mind. Machine learning was mainly a research discipline, and it was uncommon to see ML solution deployed for a business problem outside of “Big Tech” companies in the information technology industry. So

¹By *production* we understand the setting where a product or a service are made available for use by its intended audience.

the purpose of such a survey was to show that ML can be used to solve a variety of problems, and illustrate it with examples, as was done by Pěchouček and Mařík [4]. Nowadays the focus has changed: machine learning is commonly adopted in many industries, and the question becomes not “Where is it used?”, but rather “How difficult is it to use?”

A popular approach to assess the current state of machine learning deployment for businesses is a survey conducted among professionals. Such surveys are primarily undertaken by private companies, and encompass a variety of topics. Algorithmia’s report ([5], [6]) goes deep into deployment timeline, with the majority of companies reporting between 8 and 90 days to deploy a single model, and 18% taking even more time. A report by IDC [7] that surveyed 2,473 organizations and their experience with ML found that a significant portion of attempted deployments fail, quoting lack of expertise, bias in data and high costs as primary reasons. O’Reilly has conducted an interview study that focused on ML practitioners’ work experience and tools they use [8]. A broader interview-based reports have also been produced by dotscience [9] and dimensional research [10].

While there are a number of business reports on the topic, the challenges of the entire machine learning deployment pipeline are not covered nearly as widely in the academic literature. There is a burgeoning field of publications focusing on specific aspects of deployed ML, such as Bhatt et al. [11] which focuses on explainable ML or Amershi et al. [12] which discusses software engineering aspects of deploying ML. A large number of industry-specific surveys also exist, and we review some of these works in the appropriate sections below. However the only general purpose survey we found is Baier et al. [13], which combines literature review and interviews with industry partners. In contrast with that paper, which concentrates on experience of information technology sector, we aim at covering case studies from a wide variety of industries. We also discuss the most commonly reported challenges in the literature in much greater detail.

In our survey we consider three main types of papers:

- Case study papers that report experience from a single ML deployment project. Such works usually go deep into discussing each challenge the authors faced and how it was overcome.
- Review papers that describe applications of ML in a particular field or industry. These reviews normally give a summary of challenges that are most commonly encountered during the deployment of the ML solutions in the reviewed field.
- “Lessons learned” papers where authors reflect on their past experiences of deploying ML in production.

To ensure that this survey focuses on the current challenges, only papers published in the last five years are considered, with only few exceptions to this rule. We also refer other types of papers where it is appropriate, e.g. practical guidance reports, interview studies and regulations. We have not conducted any interviews ourselves as part of this study. Further work and extensions are discussed at the end of this paper.

The main contribution of our paper is to show that practitioners face challenges at each stage of the machine learning deployment workflow. This survey supports our goal to raise awareness in the academic community to the variety of problems that practitioners face when deploying machine learning, and start a discussion on what can be done to address these problems.

2 Machine Learning Deployment Workflow

For the purposes of this work we are using the ML deployment workflow definition suggested by Ashmore et al. [14], however it is possible to conduct a similar review with any other pipeline description. In this section we give a brief overview of the definition we are using.

According to Ashmore et. al. [14], the process of developing an ML-based solution in an industrial setting consists of four stages:

- **Data management**, which focuses on preparing data that is needed to build a machine learning model;
- **Model learning**, where model selection and training happens;
- **Model verification**, the main goal of which is to ensure model adheres to certain functional and performance requirements;

- **Model deployment**, which is about integration of the trained model into the software infrastructure that is necessary to run it. This stage also covers questions around model maintenance and updates.

Each of these stages is broken down further into smaller steps. It is important to highlight that the apparent sequence of this description is not necessarily the norm in a real-life scenario. It is perfectly normal for these stages to run in parallel to a certain degree and inform each other via feedback loops. Therefore this or any similar breakdown should be considered not as a timeline, but rather as a useful abstraction to simplify referring to concrete parts of the deployment pipeline.

For the remainder of this paper we discuss common issues practitioners face at each each stage and granular constituent steps. We also discuss cross-cutting aspects that can affect every stage of the deployment pipeline. Table 1 provides a summary of the issues and concerns we discuss. We do not claim that this list is exhaustive, nonetheless by providing illustrative examples for each step of the workflow we show how troublesome the whole deployment experience is today.

3 Data Management

Data is an integral part of any machine learning solution. Overall effectiveness of the solution depends on the training and test data as much as on the algorithm. The process of creating quality datasets is usually the very first stage in any production ML pipeline. Unsurprisingly, practitioners face a range of issues while working with data as previously reported by Polyzotis et al. [15]. Consequently, this stage consumes time and energy that is often not anticipated beforehand. In this section, we describe issues concerning four steps within data management: data collection, data preprocessing, data augmentation and data analysis. Note that we consider storage infrastructure challenges, such as setting up databases and query engines, beyond the scope of this survey.

3.1 Data collection

Data collection involves activities that aim to discover and understand what data is available, as well as how to organize convenient storage for it. The task of discovering what data exists and where it is can be a challenge by itself, especially in large production environments. Finding data sources and understanding their structure is a major task, which may prevent data scientists from even getting started on the actual application development. As explained by Lin and Ryaboy [16], at Twitter this situation often happened due to the fact that the same entity (e.g. a Twitter user) is processed by multiple services. Internally Twitter consists of multiple services calling each other, and every service is responsible for a single operation. This approach, known in software engineering as “single responsibility principle” [17], results in an architecture that is very flexible in terms of scalability and modification. However, the flip side of this approach is that at a large scale it is impossible to keep track of what data related to the entity is being stored by which service, and in which form. Besides, some data may only exist in a form of logs, which by their nature are not easily parsed or queried. An even worse case is the situation when data is not stored anywhere, and in order to build a dataset one needs to generate synthetic service API calls. Such a dispersion of data creates major hurdles for data scientists, because without a clear idea of what data is available or can be obtained it is often impossible to understand what ML solutions can realistically achieve.

3.2 Data preprocessing

The preprocessing step normally involves a range of data cleaning activities: imputation of missing values, reduction of data into an ordered and simplified form, and mapping from raw form into a more convenient format. Methods for carrying out data manipulations like this is an area of research that goes beyond the scope of this study. We encourage readers to refer to review papers on the topic, such as Abedjan et al. [18], Patil and Kulkarni [19], Ridzuan et al. [20]. An approach towards classifying readiness of data for ML tasks is given by Lawrence [21].

A lesser known but also important problem that can also be considered an object of the preprocessing step is data dispersion. It often turns out that there can be multiple relevant separate data sources which may have different schemas, different conventions, and their own way of storing and accessing the data. Joining this information into a single dataset suitable for machine learning can be a complicated task on its own right. An example of this is what developers of Firebird faced

Table 1: All considerations, issues and concerns explored in this study. Each is assigned to the stage and step of the deployment workflow where it is commonly encountered.

Deployment Stage	Deployment Step	Considerations, Issues and Concerns
Data management	Data collection	Data discovery
	Data preprocessing	Data dispersion Data cleaning
	Data augmentation	Labeling of large volumes of data Access to experts Lack of high-variance data
	Data analysis	Data profiling
Model learning	Model selection	Model complexity Resource-constrained environments Interpretability of the model
	Training	Computational cost Environmental impact
	Hyper-parameter selection	Resource-heavy techniques Hardware-aware optimization
Model verification	Requirement encoding	Performance metrics Business driven metrics
	Formal verification	Regulatory frameworks
	Test-based verification	Simulation-based testing
Model deployment	Integration	Operational support Reuse of code and models Software engineering anti-patterns Mixed team dynamics
	Monitoring	Feedback loops Outlier detection Custom design tooling
	Updating	Concept drift Continuous delivery
Cross-cutting aspects	Ethics	Country-level regulations Focus on technical solution only Aggravation of biases Authorship Decision making
	End users' trust	Involvement of end users User experience Explainability score
	Security	Data poisoning Model stealing Model inversion

[22]. Firebird is an advisory system in the Atlanta Fire Department, that helps identify priority targets for fire inspections. As a first step towards developing Firebird data was collected from 12 datasets including history of fire incidents, business licenses, households and more. These datasets were combined to give a single dataset covering all relevant aspects of each property monitored by the Fire Department. Authors particularly highlight data joining as a difficult problem. Given the fact that building location is the best way to identify that building, each dataset contained spatial data specifying building's address. Spatial information can be presented in different formats, and sometimes contain minor differences such as different spellings. All this needed to be cleaned and corrected, and turned out to be a massive effort that consumed a lot of time.

3.3 Data augmentation

There are multiple reasons why data might need to be augmented, and in practice one of the most problematic ones is the absence of labels. Real-world data is often unlabeled, thus labeling turns out to be a challenge in its own right. We discuss three possible factors for lack of labeled data: limited access to experts, absence of high-variance data, and sheer volume.

Labels assignment is difficult in environments that tend to generate large volumes of data, such as network traffic analysis. To illustrate a scale of this volume, a single 1-GB/s Ethernet interface can deliver up to 1.5 million packets per second. Even with a huge downsampling rate this is still a significant number, and each sampled packet needs to be traced in order to be labeled. This problem is described by Pacheco et al. [23], which surveys applications of machine learning to network traffic classification, with tasks such as protocol identification or attack detection. There are two main ways of acquiring data in this domain, and both are complicated for labeling purposes:

- Uncontrolled, collecting real traffic. This approach requires complex tracking flows belonging to a specific application. Due to this complexity very few works implement reliable ground truth assignment for real traffic.
- Controlled, emulating or generating traffic. Even in this case it has been shown that existing tools for label assignment introduce errors into collected ML datasets, going as high as almost 100% for certain applications. Moreover, these tools' performance degrades severely for encrypted traffic.

Access to experts can be another bottleneck for collecting high-quality labels. It is particularly true for areas where expertise mandated by the labeling process is significant, such as medical image analysis [24]. Normally multiple experts are asked to label a set of images, and then these labels are aggregated to ensure quality. This is rarely feasible for big datasets due to experts' availability. A possible option here is to use noisy label oracles or weaker annotations, however these approaches are by their definition a trade off that provides imprecise labels, which ultimately leads to a severe loss in quality of the model. Such losses are unacceptable in healthcare industry, where even the smallest deviation can cause catastrophic results (this is known as The Final Percent challenge).

Lack of access to high-variance data can be among the main challenges one faces when deploying machine learning solution from lab environment to real world. Dulac-Arnold et al. [25] explain that this is the case for Reinforcement Learning (RL). It is common practice in RL research to have access to separate environments for training and evaluation of an agent. However, in practice all data comes from the real system, and the agent can no longer have a separate exploration policy - this is simply unsafe. Therefore the data available becomes low-variance. While this approach ensures safety, it means that agent is not trained to recognize an unsafe situation and make right decision in it. A practical example of this issue is the goal specification for autonomous vehicles [26].

3.4 Data analysis

Data needs to be analyzed in order to uncover potential biases or unexpected distributions in it. Availability of high quality tools is essential for conducting any kind of data analysis. One area that practitioners find particularly challenging in that regard is visualization for data profiling [27]. Data profiling refers to all activities associated with troubleshooting data quality, such as missing values, inconsistent data types and verification of assumptions. Despite obvious relevance to the fields of databases and statistics, there are still too few tools that enable efficient execution of these data mining tasks. The need for such tools becomes apparent considering that, according to the

survey conducted by Microsoft [28], data scientists think about data issues as the main reason to doubt quality of the overall work.

4 Model Learning

Model learning is the stage of the deployment workflow that enjoys the most attention within the academic community. All modern research in machine learning methods contributes towards better selection and variety of models and approaches that can be employed at this stage. As an illustration of the scale of the field’s growth, the number of submissions to NeurIPS, primary conference on ML methods, has quadrupled in six years, going from 1678 submissions in 2014 to 6743 in 2019 [29]. Nevertheless, there is still plenty of practical considerations that affect the model learning stage. In this section, we discuss issues concerning three steps within model learning: model selection, training and hyper-parameter selection.

4.1 Model selection

In many practical cases the selection of a model is often decided by one key characteristic of a model: complexity. Despite areas such as deep learning and reinforcement learning gaining increasing levels of popularity with the research community, in practice simpler models are often chosen as we explain below. Such model include shallow network architectures, simple PCA-base approaches, decision trees and random forests.

Simple models can be used as a way to prove the concept of the proposed ML solution and get the end-to-end setup in place. This approach accelerates the time to get a deployed solution, allows the collection of important feedback and also helps avoid overcomplicated designs. This was the case reported by Haldar et al. [30]. In the process of applying machine learning to AirBnB search, the team started with a complex deep learning model. The team was quickly overwhelmed by its complexity and ended up consuming development cycles. After several failed deployment attempts the neural network architecture was drastically simplified: a single hidden layer NN with 32 fully connected ReLU activations. Even such a simple model had value, as it allowed the building of a whole pipeline of deploying ML models in production setting, while providing reasonably good performance². Over time the model evolved, with a second hidden layer being added, but it still remained fairly simple, never reaching the initially intended level of complexity.

Another advantage that less complex models can offer is their relatively modest hardware requirements. This becomes a key decision point in resource constrained environments, as shown by Wagstaff et al. [31]. They worked on deploying ML models to a range of scientific instruments onboard Europa Clipper spacecraft. Spacecraft design is always a trade-off between the total weight, robustness and the number of scientific tools onboard. Therefore computational resources are scarce and their usage has to be as small as possible. These requirements naturally favor the models that are light on computational demands. The team behind Europa Clipper used machine learning for three anomaly detection tasks, some models took time series data as input and some models took images, and on all three occasions simple threshold or PCA based techniques were implemented. They were specifically chosen because of their robust performance and low demand on computational power.

A further example of a resource-constrained environment is wireless cellular networks, where energy, memory consumption and data transmission are very limited. Most advanced techniques, such as deep learning, are not considered yet for practical deployment, despite being able to handle highly dimensional mobile network data [32].

The ability to interpret the output of a model into understandable business domain terms often plays a critical role in model selection, and can even outweigh performance considerations. For that reason decision trees (DT), which can be considered a fairly basic ML algorithm, are widely used in practice. For example, Hansson et al. [33] describe several cases in manufacturing that adopt DT due to its high interpretability.

Banking is yet another example of an industry where DT finds extensive use. As an illustrative example, it is used by Keramati et al. [34] where the primary goal of the ML application is to predict customer churn by understanding if-then rules. While it is easy to imagine more complicated

²We discuss more benefits of setting up the automated deployment pipeline in Section 6.3.

models learning the eventual input-output relationship for this specific problem, interpretability is key requirement here because of the need to identify the features of churners. The authors found DT to be the best model to fulfill this requirement.

Nevertheless, deep learning (DL) is commonly used for practical background tasks that require analysis a large amount of previously acquired data. This notion is exemplified by the field of unmanned aerial vehicles (UAV) [35]. Image sensors are commonplace in UAVs due to their low cost, low weight, and low power consumption. Consequently, processing images acquired from sensors is the main way of exploiting excellent capabilities in processing and presentation of raw data that DL offers. But computational resource demands still remain the main blocker for deploying DL as an online processing instrument on board of UAVs.

4.2 Training

One of the biggest concern with model training is the economic cost associated with carrying the training stage due to the computational resources required. This is certainly true in the field of natural language processing (NLP), as illustrated by Sharir et al. [36]. The authors observe that while the cost of individual floating-point operations is decreasing, the overall cost of training NLP is only growing. They took one of the state-of-the-art models in the field, BERT [37], and found out that depending on the chosen model size full training procedure can cost anywhere between \$50k and \$1.6m, which is unaffordable for most research institutions and even companies. The authors observe that training dataset size, number of model parameters and number of operations utilized by the training procedure are all contributing towards the overall cost. Of particular importance here is the second factor: novel NLP models are already using billions of parameters, and this number is expected to increase further in the nearest future [38].

A related concern is raised by Strubell et al. [39] regarding the impact the training of ML models has on the environment. By consuming more and more computational resources, ML models training is driving up the energy consumption and greenhouse gas emissions. According to the estimates provided in the paper, one full training cycle utilizing neural architecture search emits the amount of CO₂ comparable to what four average cars emit in their whole lifetime. The authors stress how important it is for researchers to be aware of such impact of model training, and argue that community should give higher priority to computationally efficient hardware and algorithms.

4.3 Hyper-parameter selection

In addition to parameters that are learned during the training process, many ML models also define several hyper-parameters. Hyper-parameter optimization (HPO) is the process of choosing the optimal set of these hyper-parameters. Most HPO techniques involve multiple training cycles of the ML model. Besides, the size of HPO task grows exponentially with each new hyper-parameter, because it adds a new dimension to the search space. As discussed by Yang and Shami [40], these considerations make HPO techniques very expensive and resource-heavy in practice, especially for applications of deep learning. Even approaches like Hyperband [41] or Bayesian optimization [42], that are specifically designed to minimize the number of training cycles needed, are still not able to deal with certain problems due to the complexity of the models or the size of the datasets.

HPO often needs to take into account specific requirements imposed by the environment where the model will run. This is exemplified by Marculescu et al. [43] in the context of hardware-aware ML. In order to deploy models to embedded and mobile devices, one needs to be well aware of energy and memory constraints imposed by such devices. This creates a need for customized hardware-aware optimization techniques that co-optimize for the accuracy of the model and the hardware efficiency.

5 Model Verification

The goal of the model verification stage is multifaceted, because an ML model should generalize well to unseen inputs, demonstrate reasonable handling of edge cases and overall robustness, as well as satisfy all functional requirements. In this section, we discuss issues concerning three steps within model verification: requirement encoding, formal verification and test-based verification.

5.1 Requirement encoding

Defining requirements for a machine learning model is a crucial prerequisite of testing activities. It often turns out that an increase in model performance does not translate into a gain in business value, as Booking.com discovered after deploying 150 models into production [44]. Therefore more specific metrics need to be defined and measured, such as KPIs and other business driven measures. In the case of Booking.com such metrics included conversion, customer service tickets or cancellations. Cross-disciplinary effort is needed to even define such metrics, as understanding from modeling, engineering and business angles is required. Once defined, these metrics are used for monitoring of the production environment and for quality control of model updates.

Besides, simply measuring the accuracy of the ML model is not enough to understand its performance. Essentially, performance metrics should reflect audience priorities. For instance Sato et al. [45] recommend validating models for bias and fairness, while in the case described by Wagstaff et al. [31] controlling for consumption of spacecraft resources is crucial.

5.2 Formal Verification

The formal verification step verifies that the model functionality follows the requirements defined within the scope of the project. Such verification could include mathematical proofs of correctness or numerical estimates of output error bounds, but as Ashmore et. al. [14] point out this rarely happens in practice. More often quality standards are being formally set via extensive regulatory frameworks.

An example of where ML solutions have to adhere to regulations is the banking industry [46]. This requirement was developed in the aftermath of the global financial crisis, as the industry realized that there was a need for heightened scrutiny towards models. As a consequence an increased level of regulatory control is now being applied to the processes that define how the models are built, approved and maintained. For instance, official guidelines has been published by the UK's Prudential Regulation Authority [47] and European Central Bank [48]. These guidelines require model risk frameworks to be in place for all business decision-making solutions, and implementation of such frameworks requires developers to have extensive tests suites in order to understand behavior of their ML models. The formal verification step in that context means ensuring that the model meets all criteria set by the corresponding regulations.

Regulatory frameworks share similarities with country-wide policies, which we discuss in greater details in Section 7.1.

5.3 Test-based Verification

Test-based verification is intended for ensuring that the model generalizes well to the previously unseen data. While collecting validation dataset is usually not a problem, as it can be derived from splitting the training dataset, it may not be enough for production deployment.

In an ideal scenario testing is done in a real-life setting, where business driven metrics can be observed, as we discussed in Section 5.1. Full scale testing in real-world environment can be challenging for a variety of safety, security and scale reasons, and is often substituted with testing in simulation. That is the case for models for autonomous vehicles control [26]. Simulations are cheaper, faster to run, and provide flexibility to create situations rarely encountered in real life. Thanks to these advantages, simulations are becoming prevalent in this field. However, it is important to remember that simulation-based testing hinges on assumptions made by simulation developers, and therefore cannot be considered a full replacement for real-world testing. Even small variations between simulation and real world can have drastic effects on the system behavior, and therefore the authors conclude that validation of the model and simulation environment alone is not enough for autonomous vehicles. This point is emphasized further by the experiences from the field of reinforcement learning [25], where use of simulations is a de-facto standard for training agents.

In addition, the dataset itself also needs to be constantly validated to ensure data errors do not creep into the pipeline and do not affect the overall quality. Breck et al. [49] argue that one of the most common scenarios when issues in data can go unnoticed is the setup where data generation is decoupled from the ML pipeline. There could be multiple reasons for such issues to appear, including bugs in code, feedback loops, changes in data dependencies. Data errors can propagate

and manifest themselves at different stages of the pipeline, therefore it is imperative to catch them early by including data validation routines into the ML pipeline.

6 Model Deployment

Machine learning systems running in production are complex software systems that have to be maintained over time. This presents developers with another set of challenges, some of which are shared with running regular software services, and some are unique to ML.

There is a separate discipline in engineering, called DevOps, that focuses on techniques and tools required to successfully maintain and support existing production systems. Consequently, there is a necessity to apply DevOps principles to ML systems. However, even though some of the DevOps principles apply directly, there is also a number of challenges unique to productionizing machine learning. This is discussed in detail by Dang et al. [50] which uses the term AIOps for DevOps tasks for ML systems. Some of the challenges mentioned include lack of high quality telemetry data as well as no standard way to collect it, difficulty in acquiring labels which makes supervised learning approaches inapplicable³ and lack of agreed best practices around handling of machine learning models. In this section, we discuss issues concerning three steps within model deployment: integration, monitoring and updating.

6.1 Integration

The model integration step constitutes of two main activities: building the infrastructure to run the model and implementing the model itself in a form that can be consumed and supported. While the former is a topic that belongs almost entirely in systems engineering and therefore lies out of scope of this work, the latter is of interest for our study, as it exposes important aspects at the intersection of ML and software engineering. In fact, many concepts that are routinely used in software engineering are now being reinvented in the ML context.

Code reuse is a common topic in software engineering, and ML can benefit from adopting the same mindset. Reuse of data and models can directly translate into savings in terms of time, effort or infrastructure. An illustrative case is the approach Pinterest took towards learning image embeddings [51]. There are three models used in Pinterest internally which use similar embeddings, and initially they were maintained completely separately, in order to make it possible to iterate on the models individually. However, this created engineering challenges, as every effort in working with these embeddings had to be multiplied by three. Therefore the team decided to investigate the possibility of learning universal set of embeddings. It turned out to be possible, and this reuse ended up simplifying their deployment pipelines as well as improving performance on individual tasks.

A broad selection of engineering problems that machine learning practitioners now face is given in Sculley et al. [52]. Most of them are considered anti-patterns in engineering, but are currently widespread in machine learning software. Some of these issues, such as abstraction boundaries erosion and correction cascades, are caused by the fact that ML is used in cases where the software has to take explicit dependency on external data. Others, such as glue code or pipeline jungles, stem from the general tendency in the field to develop general-purpose software packages. Yet another source of problems discussed in the paper is the configuration debt, which is caused by the fact that ML systems, besides all configurations a regular software system may require, add a sizable number of ML-specific configuration settings that have to be set and maintained.

Researchers and software engineers often find themselves working together on the same project aiming to reach a business goal with a machine learning approach. On surface there seems to be a clear separation of responsibilities: researchers produce the model while engineers build infrastructure to run it. In reality, their areas of concern often overlap when considering the development process, model inputs and outputs and performance metrics. Contributors in both roles often work on the same code. Thus it is beneficial to loop researchers into the whole development journey, making sure they own the product code base along with the engineers, use the same version control and participate in code reviews. Despite obvious onboarding and slow-start challenges, this approach was seen to bring long term benefits in terms of speed and quality of product delivery [12].

³Please refer to Section 3.3 for detailed discussion about data labeling.

6.2 Monitoring

Monitoring is one of the issues associated with maintaining machine learning systems as reported by Sculley et al. [52]. The community is in the early stages of understanding what are the key metrics of data and models to monitor and how to alarm on them. Monitoring of evolving input data, prediction bias and overall performance of ML models is an open problem. Another maintenance issue highlighted by this paper that is specific to data-driven decision making is feedback loops. ML models in production can influence their own behavior over time via regular retraining. While making sure the model stays up to date, it is possible to create feedback loop where the input to the model is being adjusted to influence its behavior. This can be done intentionally, as well as happen inadvertently which is a unique challenge when running live ML systems.

Klaise et al. [53] point out the importance of outlier detection as a key instrument to flag model predictions that cannot be used in a production setting. The authors name two reasons for such predictions to occur: the inability of the models to generalize outside of the training dataset and also overconfident predictions on out-of-distribution instances due to poor calibration. Deployment of the outlier detector can be a challenge in its own right, because labeled outlier data is scarce, and the detector training often becomes a semi-supervised or even an unsupervised problem.

Additional insight on monitoring of ML systems can be found in Ackermann et al. [54]. This paper describes an early intervention system (EIS) for two police departments in the US. On the surface their monitoring objectives seem completely standard: data integrity checks, anomaly detection and performance metrics. One would expect to be able to use out-of-the-box tooling for these tasks. However, the authors explain that they had to build all these checks from scratch in order to maintain good model performance. For instance, the data integrity check meant verifying updates of a certain input table and checksums on historical records, performance metric was defined in terms of the number of changes in top k outputs, and anomalies were tracked on rank-order correlations over time. All of these monitoring tools required considerable investigation and implementation. This experience report highlights a common problem with currently available end-to-end ML platforms: the final ML solutions are usually so sensitive to problem's specifics that out-of-the-box tooling does not fit their needs well.

As a final remark we note that there is an overlap between choice of metrics for monitoring and validation. The latter topic is discussed in Section 5.1.

6.3 Updating

Once the initial deployment of the model is completed, it is often necessary to be able to update the model later on in order to make sure it always reflects the most recent trends in data and the environment. There are multiple techniques for adapting models to a new data, including scheduled regular retraining and continual learning [55]. Nevertheless in production setting model updating is also affected by practical considerations.

A particularly important problem that directly impacts the quality and frequency of model update procedure is the concept drift. Concept drift in ML is understood as changes observed in joint distribution $p(X, y)$, where X is the model input and y is the model output. Undetected, this phenomenon can have major adverse effects on model performance, as is shown by Jameel et al. [56] for classification problems or by Celik and Vanschoren [57] in AutoML context. Concept drift can arise due to a wide variety of reasons. For example, the finance industry faced turbulent changes as the financial crisis of 2008 was unfolding, and if advanced detection techniques were employed it could have provided additional insights into the ongoing crisis, as explained by Masegosa et al. [58]. Changes in data can also be caused by inability to avoid fluctuations in the data collection procedure, as described in paper Langenkämper et al. [59] which studies the effects of slight changes in marine images capturing gear and location on deep learning models' performance. Data shifts can have noticeable consequences even when occurring at microscopic scale, as Zenisek et al. [60] show in their research on predictive maintenance for wear and tear of industrial machinery. Even though concept drift has been known for decades [61], these examples show that it remains a critical problem for applications of ML today.

On top of the question of when to retrain the model to keep it up to date, there is an infrastructural question on how to deliver the model artifact to the production environment. In software engineering such tasks are commonly solved with continuous delivery (CD), which is an approach for

accelerating development cycle by building an automated pipeline for building, testing and deploying software changes. CD for machine learning solutions is complicated because, unlike in regular software products where changes only happen in the code, ML solutions experience change along three axis: the code, the model and the data. An attempt to formulate CD for ML as a separate discipline can be seen in Sato et al. [45]. This work describes the pieces involved and the tools that can be used at each step of building the full pipeline. A direct illustration of benefits that a full CD pipeline can bring to the real-life ML solution can be found in Wider and Deger [62].

7 Cross-cutting aspects

In this section we describe three additional aspects that ML projects have to consider: ethics, end users' trust and security. These aspects can affect every stage of the deployment pipeline.

7.1 Ethics

Ethical considerations should always inform data collection activities. As stated in the report on ethical AI produced by the Alan Turing Institute [63], “it is essential to establish a continuous chain of human responsibility across the whole AI project delivery workflow”. If researchers and developers do not follow this recommendation, complications may come up due to a variety of reasons: breach of governmental regulations, unjustifiable outcomes, aggravation of existing issues, and more [63].

Various countries have produced regulations to protect personal data rights. The more sensitive is the information collected from the individual, the severer are the regulations governing its use. And of course industry that deals with some of the most sensitive information is healthcare. According to Han et al. [64], many countries have strict laws in place to protect data of patients, which makes adoption of ML in healthcare particularly difficult. Examples of such regulations include the General Data Protection Regulation in European Union [65] and ethical screening laws in a range of Asian countries [66]. On one hand there is no doubt that these rules are absolutely necessary to make sure people are comfortable with their data being used. On the other hand amount of reviews, software updates and cycles of data collection/annotation that is required makes it exceptionally hard to keep up with technical advances in ML, as Han et al. [64] explain following their experience deploying ML solutions in healthcare sector in Japan.

Companies should not be focusing solely on the technological side of their solutions, as DeepMind and Royal Free NHS Foundation Trust have discovered while working on Streams, an application for automatic review of test results for serious conditions [67]. Their initial collaboration was not specific enough on the use of patient data and on patient involvement overall, which triggered an investigation on their compliance with data protection regulations. The revised collaboration agreement was far more comprehensive and included patient and public engagement strategy in order to ensure data is being used ethically.

Since ML models use previously seen data to make decisions, they can worsen issues that already exist in data. This effect in the field of criminal justice is discussed in detail by O’Neil [68]. Models that calculate person’s criminal “risk score” are often marketed as a way to remove human bias. Nevertheless, they use seemingly neutral demographic information that often ends up serving as a proxy. As a result, people are disadvantaged on the basis of race or income.

Likewise, Soden et al. [69] mention aggravation of social inequalities through use of biased training datasets as one of the main concerns in applying ML to Disaster Risk Management (DRM) field. Furthermore, it is argued that ML causes privacy and security concerns in Fragility, Conflict and Violence settings through combination of previously distinct datasets. Reducing role of both experts and general public is also seen as an ethical issue by DRM professionals. Some of these issues are studied by the branch of machine learning known as Fairness in ML [70]. We discuss related cross-cutting security concerns in Section 7.3.

An interesting ethical aspect arises in usage of ML in the field of creative arts, discussed by Anantrasirichai and Bull [71]. When a trained model is used to create a piece of visual art, it is not entirely clear where the authorship of this piece resides. The questions of originality therefore requires a special attention. Closely related with this question is the growing concern of fake content being generated with ML, which can be easily used for the wrong purposes [72].

Another facet of ethical issues encountered by machine learning is the data based decision making. As machine learning tools become utilized in critical decision making processes ethical concerns with their usage grow. Illustrative note is made by Muthiah et al. [73]. Their system of predicting civil unrest, called EMBERS, is designed to be used as a forecasting and communication tool, however authors remark that it can also be potentially misused by governments, either due to misunderstanding of its role in society, or intentionally.

7.2 End users' trust

ML is often met cautiously by the end users [74], [3], [75]. On their own accord models provide minimal explanations, which makes it difficult to persuade end users in their utility [64]. In order to convince users to trust ML based solutions, time has to be invested to build that trust. In this section, we explore ways in which that is done in practice.

If an application has a well-defined accessible audience, getting these people involved early in the project is an efficient way to foster their confidence in the end product. This approach is very common in medicine, because the end product is often targeted at a well defined group of healthcare workers. One example is the project called Sepsis Watch [76]. In this project the goal was to build a model that estimates patient's risk of developing sepsis. It was not the first attempt at automating this prediction, and since previous attempts were considered failures, medical personnel were skeptical about eventual success of Sepsis Watch. To overcome this skepticism, the team prioritized building trust, by:

- Building strong communication channels;
- Sharing with stakeholders progress towards developing the goal instead of showing technical advances;
- Establishing mechanisms for public and external accountability;
- Engaging both front-line clinicians and enterprise-level decision makers at early stages of the project.

One of the key messages of this work is that model interpretability has limits as a trust-building tool, and other ways to achieve high credibility with the end users should be considered. While it may sound controversial, in fact it aligns with conclusions made by "Project explAIn" which found that relative importance of explanations of AI decisions varies by context [77].

Similar argument is made by Soden et al. [69], who explore an impact ML has on disaster risk management (DRM). Due to growing complexity of the ML solutions deployed, it is becoming increasingly hard for public to participate and consequently to trust the ML-based DRM services, such as flooding area estimates or prediction of damage from a hurricane. As a mitigation measure the authors recommend making development of these solutions as transparent as possible, by taking into account voice of residents in the areas portrayed by models as "at risk" and relying on open software and data whenever possible.

A reasonable approach towards building trustworthy products is investing time in specialised user interfaces with tailored user experience. Developers of Firebird [22], a system that helps identify priority targets for fire inspection in the city of Atlanta, USA, found that the best way to introduce ML solution as a replacement of the previously used pen-and-paper method was to develop a user interface that presented the results of modelling in a way that the end users (fire officers and inspectors in the Fire dept) found most useful and clear. Similar experience is reported by Ackermann et al. [54].

Authors of EMBERS [73], a system that forecasts population-level events (such as protest), in Latin America, noticed that their users have two modes of using the system: (a) high recall: obtain most events, and then filter them using other methods; (b) high precision: focus on specific area or specific hypothesis. To improve the user experience and thus increase their confidence in the product, user interface was improved to easily support both modes. This case study emphasizes the importance of context-aware personalization for ML systems' interfaces, one of the key observations delivered by Project explAIn [77].

Budd et al. [24] show that interface design directly impacts quality of applications built to collect annotations for unlabeled data. They discuss a range of projects that collected labels for medical

images, all of which benefited from well designed user interface. The authors conclude that end user interface plays a large part in overall success of the annotation applications.

Finally, the solutions based on models whose decisions can be explained are preferred when the target audience has experience and an understanding of ML. Bhatt et al. [11] analyzed explainability as a feature of machine learning models deployed within enterprises, and found that it is a must-have requirement for most of the stakeholders, including executives, ML engineers, regulators, and others. Moreover, their survey showed that explainability score is a desired model metric, along with measures of fairness and robustness.

7.3 Security

Machine Learning opens up new threat vectors across the whole ML deployment workflow as described by Kumar et al. [78]. Specialised adversarial attacks for ML can occur on the model itself, the data used for training but also the resulting predictions. The field of adversarial machine learning studies the effect of such attacks against ML models and how to protect against them [79, 80]. Recent work from Kumar et al. found that industry practitioners are not equipped to protect, detect and respond to attacks on their ML systems [81]. In this section, we describe the three most common attacks reported in practice which affects deployed ML models: data poisoning, model stealing and model inversion. We focus specifically on adversarial machine learning and consider other related general security concerns in deploying systems such as access control and code vulnerabilities beyond the scope of our work.

In data poisoning, the goal of the adversarial attack is to deliberately corrupt the integrity of the model during the training phase in order to manipulate the produced results. Poisoning attacks are particularly relevant in situations where the machine learning model is continuously updated with new incoming training data. Jagielski et al. reported that in a medical setting using a linear model, the introduction of specific malicious samples with a 8% poisoning rate in the training set resulted in incorrect dosage for half of the patients [82].

Data poisoning can also occur as a result of a coordinated collective effort that exploits feedback loops we have discussed in Section 6.2, as it happened with Microsoft’s Twitter bot Tay [83]. Tay was designed to improve its understanding of the language over time, but was quickly inundated with a large number of deliberately malevolent tweets. Within 16 hours of its release a troubling percentage of Tay’s messages were abusive or offensive, and the bot was taken down.

Another type of adversarial attack is reverse engineering a deployed model by querying its inputs (e.g. via a public prediction API) and monitoring the outputs. The adversarial queries are crafted to maximize the extraction of information about the model in order to train a substitute model. This type of attack is referred to as model stealing. In a nutshell, this attack results in loss of intellectual property which could be a key business advantage for the defender. Tramèr et al. [84] have shown that it is possible to replicate models deployed in production from ML services offered by Google, Amazon and Microsoft across a range of ML algorithms including logistic regression, decision trees, SVMs and neural networks. In their work, they report the number of queries ranging from 650 to 4013 to extract an equivalent model and in time ranging from 70s to 2088s.

A related attack is that of model inversion where the goal of the adversarial attack is recover parts of the private training set, thereby breaking its confidentiality. Fredrikson et al. have shown that they could recover training data by exploiting models that report confidence values along their predictions [85]. Veale et al. [86] emphasize the importance of protecting against model inversion attacks as a critical step towards compliance with data protection laws such as GDPR.

8 Discussion of potential solutions

This survey looked at case studies from a variety of industries: computer networks, manufacturing, space exploration, law enforcement, banking, and more. However, further growths of ML adoption can be severely hindered by poor deployment experience. To make the ML deployment scalable and accessible to every business that may benefit from it, it is important to understand the most critical pain points and to provide tools, services and best practices that address those points. We see this survey as an initial step in this direction: by recognizing the most common challenges currently being reported we hope to foster an active discussion within the academic community about what

possible solutions might be. We classify possible research avenues for solutions into two categories, which we discuss below.

8.1 Tools and services

The market for machine learning tools and services is experiencing rapid growth [87]. As a result, tools for individual deployment problems are continuously developed and released. For some of the problems we have highlighted, making use of the right specific tool is an entirely reasonable approach.

For example, this is most likely the case for operational maintenance of ML models. Many platforms on the market offer end-to-end experience for the user, taking care of such things as data storage, retraining and deployment. Examples include AWS SageMaker [88], Microsoft AzureML [89], Uber Michelangelo [90], TensorFlow TFX [91], MLflow [92] and more. A typical ML platform would include, among other features, data storage facility, model hosting with APIs for training and inference operations, a set of common metrics to monitor model health and an interface to accept custom changes from the user. By offering managed infrastructure and a range of out-of-the-box implementations for common tasks such platforms greatly reduce operational burden associated with maintaining the ML model in production.

Quality assurance also looks to be an area where better tools can be of much assistance. As mentioned in Section 3, data integrity plays a big part in quality control, and is an active branch of research [18]. Models themselves can also greatly benefit from development of a test suite to verify their behavior. This is normally done by trial and error, but more formal approaches are being developed, as is the case with CheckList methodology for NLP models [93].

As discussed in Section 3.3, obtaining labels is often a problem with real world data. Weak supervision has emerged as a separate field of ML which looks for ways to address this challenge. Consequently, a number of weak supervision libraries are now actively used within the community, and show promising results in industrial applications. Some of the most popular tools include Snorkel [94], Snuba [95] and cleanlab [96].

Using specific tools for solving individual problems is a straightforward approach. However practitioners need to be aware that by using a particular tool they introduce an additional dependency into their solution. While a single additional dependency seems manageable, their number can quickly grow and become a maintenance burden. Besides, as we mentioned above, new tools for ML are being released constantly, thus presenting practitioners with the dilemma of choosing the right tool by learning its strength and shortcomings.

8.2 Holistic approaches

Even though ML deployments require a certain amount of software development, ML projects are fundamentally different from traditional software engineering projects, and we argue they need a different approach. The main differences arise from unique activities like data discovery, dataset preparation, model training, deployment success measurement etc. Some of these activities cannot be defined precisely enough to have a reliable time estimate, some assume huge potential risks, and some make it difficult to measure the overall added value of the project. Therefore ML deployments do not lend themselves well to widespread approaches to software engineering management paradigms, and neither to common software architectural patterns.

Data Oriented Architectures (DOA, [97], [98]) is an example of an idea that suggests rethinking how things are normally approached in software development, and by doing so promises to solve quite a few issues we have discussed in this survey. Specifically, the idea behind DOA is to consider replacing micro-service architecture, widespread in current enterprise systems, with streaming-based architecture, thus making data flowing between elements of business logic more explicit and accessible. Micro-service architectures have been successful in supporting high scalability and embracing the single responsibility principle. However, they also make data flows hard to trace, and it is up to owners of every individual service to make sure inputs and outputs are being stored in a consistent form. DOA provides a solution to this problem by moving data to streams flowing between stateless execution nodes, thus making data available and traceable by design, therefore making simpler the tasks of data discovery, collection and labeling. In its essence DOA proposes to acknowledge

that modern systems are often data-driven, and therefore need to prioritize data in their architectural principles.

As noted above, ML projects normally do not fit well with commonly used management processes, such as Scrum or Waterfall. Therefore it makes sense to consider processes tailored specifically for ML. One such attempt is done by Lavin et. al. [99], who propose Technology Readiness Levels for ML (TRL4ML) framework. TRL4ML describes a process of producing robust ML systems that takes into account key differences between ML and traditional software engineering.

A very widespread practice in software engineering is to define a set of guidelines and best practices to help developers make decisions at various stages of the development process. These guidelines can cover a wide range of questions, from variable names to execution environment setup. Similarly, Zinkevich [100] compiled a collection of best practices for machine learning that are utilized in Google. While this cannot be viewed as a coherent paradigm towards doing ML deployment, this document gives practical advice on a variety of important aspects that draw from the real life experiences of engineers and researchers in the company.

Holistic approaches are created with ML application in mind, and therefore they have the potential to offer a significant ease of deploying ML. But it should be noted that all such approaches assume a big time investment, because they represent significant changes to current norms in project management and development. Therefore a careful assessment of risks versus benefits should be carried out before adopting any of them.

9 Further Work

Even though the set of challenges we reviewed covers every stage of the ML deployment workflow, we believe that it is far from being complete. Identifying other, especially non-technical, challenges is a natural direction of further work and could be augmented by conducting interviews with industry representatives about their experiences of deploying ML.

In this paper, we reviewed reports from a variety of industries, which shows the ubiquity and variety of challenges with deploying ML in production. An interesting extension can be the comparative analysis of industries. Quantitative and qualitative analysis of most commonly reported challenges may open interesting transferability opportunities, as approaches developed in one field may be applicable in the other.

Our work includes a brief discussion of existing tools in Section 8.1. However, the community would benefit from a comprehensive review of currently available tools and services, mapped to challenges reported in our study. This new work could be combined with our survey to enable practitioners to identify the problem they are facing and choose the most appropriate tool that addresses that problem.

10 Conclusion

In this survey, we showed that practitioners deal with challenges at every step of the ML deployment workflow due to practical considerations of deploying ML in production. We discussed challenges that arise during the data management, model learning, model verification and model deployment stages, as well as considerations that affect the whole deployment pipeline including ethics, end users' trust and security. We illustrated each stage with examples across different fields and industries by reviewing case studies, experience reports and the academic literature.

We argue that it is worth academic community's time and focus to think about these problems, rather than expect each applied field to figure out their own approaches. We believe that ML researchers can drive improvements to the ML deployment experience by exploring holistic approaches and taking into account practical considerations.

As an observation that follows from the process of collecting papers to review in this survey, we note the shortage of deployment experience reports in the academic literature. Valuable knowledge obtained by industry ML practitioners goes unpublished. We would like to encourage organisations to prioritize sharing such reports, as they provide valuable information for the wider community, but also as a way to self-reflect, collect feedback and improve on their own solutions.

We hope this survey will encourage discussions within the academic community about pragmatic approaches to deploying ML in production.

Acknowledgments and Disclosure of Funding

We would like to thank our reviewers for their thoughtful comments and suggestions. We are also grateful to Jessica Montgomery, Diana Robinson and Ferenc Huszar for discussions that helped shape this work.

References

- [1] Arif Cam, Michael Chui, and Bryce Hall. Global AI survey: AI proves its worth, but few scale impact. *McKinsey Analytics*, 2019.
- [2] Thomas H. Davenport and Rajeev Ronanki. Artificial intelligence for the real world. *Harvard business review*, 96(1):108–116, 2018.
- [3] Royal Society (Great Britain). *Machine Learning: The Power and Promise of Computers that Learn by Example: an Introduction*. Royal Society, 2017.
- [4] Michal Pěchouček and Vladimír Mařík. Industrial deployment of multi-agent technologies: review and selected case studies. *Autonomous agents and multi-agent systems*, 17(3):397–431, 2008.
- [5] Kyle Wiggers. Algorithmia: 50% of companies spend between 8 and 90 days deploying a single AI model, 2019. Available at <https://venturebeat.com/2019/12/11/algorithmia-50-of-companies-spend-upwards-of-three-months-deploying-a-single-ai-model/>
- [6] Lawrence E. Hecht. Add it up: How long does a machine learning deployment take?, 2019. Available at <https://thenewstack.io/add-it-up-how-long-does-a-machine-learning-deployment-take/>
- [7] Kyle Wiggers. IDC: For 1 in 4 companies, half of all AI projects fail, 2019. Available at <https://venturebeat.com/2019/07/08/idc-for-1-in-4-companies-half-of-all-ai-projects-fail/>
- [8] Ben Lorica and Nathan Paco. *The State of Machine Learning Adoption in the Enterprise*. O'Reilly Media, 2018.
- [9] The state of development and operations of AI applications. *Dotscale*, 2019. Available at https://dotscale.com/assets/downloads/Dotscale_Survey_Report-2019.pdf
- [10] Artificial intelligence and machine learning projects are obstructed by data issues. *dimensional research*, 2019. Available at <https://telecomreseller.com/wp-content/uploads/2019/05/EMBARGOED-UNTIL-800-AM-ET-0523-Dimen>
- [11] Umang Bhatt, Alice Xiang, Shubham Sharma, Adrian Weller, Ankur Taly, Yunhan Jia, Joydeep Ghosh, Ruchir Puri, José M. F. Moura, and Peter Eckersley. Explainable machine learning in deployment, 2019.
- [12] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 291–300. IEEE, 2019.
- [13] Lucas Baier, Fabian Jöhren, and Stefan Seebacher. Challenges in the deployment and operation of machine learning in practice. 2019.
- [14] Rob Ashmore, Radu Calinescu, and Colin Paterson. Assuring the machine learning lifecycle: Desiderata, methods, and challenges. *arXiv preprint arXiv:1905.04223*, 2019.
- [15] Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. Data lifecycle challenges in production machine learning: a survey. *ACM SIGMOD Record*, 47(2):17–28, 2018.
- [16] Jimmy Lin and Dmitriy Ryaboy. Scaling big data mining infrastructure: the Twitter experience. *Acm SIGKDD Explorations Newsletter*, 14(2):6–19, 2013.

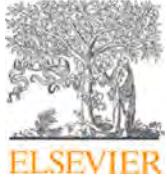
- [17] Robert C. Martin. The single responsibility principle. *The principles, patterns, and practices of Agile Software Development*, pages 149–154, 2002.
- [18] Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. Detecting data errors: Where are we and what needs to be done? *Proceedings of the VLDB Endowment*, 9(12):993–1004, 2016.
- [19] Rajashree Y Patil and RV Kulkarni. A review of data cleaning algorithms for data warehouse systems. *International Journal of Computer Science and Information Technologies*, 3(5):5212–5214, 2012.
- [20] Fakhitah Ridzuan and Wan Mohd Nazmee Wan Zainon. A review on data cleansing methods for big data. *Procedia Computer Science*, 161:731–738, 2019.
- [21] Neil D Lawrence. Data readiness levels. *arXiv preprint arXiv:1705.02245*, 2017.
- [22] Michael Madaio, Shang-Tse Chen, Oliver L. Haimson, Wenwen Zhang, Xiang Cheng, Matthew Hinds-Aldrich, Duen Horng Chau, and Bistra Dilkina. Firebird: Predicting fire risk and prioritizing fire inspections in Atlanta. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 185–194, 2016.
- [23] F. Pacheco, E. Exposito, M. Gineste, C. Baudoin, and J. Aguilar. Towards the deployment of machine learning solutions in network traffic classification: A systematic survey. *IEEE Communications Surveys Tutorials*, 21(2):1988–2014, 2019.
- [24] Samuel Budd, Emma C. Robinson, and Bernhard Kainz. A survey on active learning and human-in-the-loop deep learning for medical image analysis, 2019.
- [25] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning, 2019.
- [26] Sampo Kuutti, Richard Bowden, Yaochu Jin, Phil Barber, and Saber Fallah. A survey of deep learning applications to autonomous vehicle control, 2019.
- [27] Sean Kandel, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. Enterprise data analysis and visualization: An interview study. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2917–2926, 2012.
- [28] Miryung Kim, Thomas Zimmermann, Robert DeLine, and Andrew Begel. Data scientists in software teams: State of the art and challenges. *IEEE Transactions on Software Engineering*, 44(11):1024–1038, 2017.
- [29] Diego Charrez. NeurIPS 2019 stats, 2019. Available at <https://medium.com/@dcharrezt/neurips-2019-stats-c91346d31c8f>.
- [30] Malay Haldar, Mustafa Abdool, Prashant Ramanathan, Tao Xu, Shulin Yang, Huizhong Duan, Qing Zhang, Nick Barrow-Williams, Bradley C. Turnbull, Brendan M. Collins, and et al. Applying deep learning to AirBnB search. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Jul 2019.
- [31] Kiri L. Wagstaff, Gary Doran, Ashley Davies, Saadat Anwar, Srija Chakraborty, Marissa Cameron, Ingrid Daubar, and Cynthia Phillips. Enabling onboard detection of events of scientific interest for the Europa Clipper spacecraft. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’19, page 2191–2201, New York, NY, USA, 2019. Association for Computing Machinery.
- [32] Ursula Challita, Henrik A. Ryden, and Hugo Tullberg. When machine learning meets wireless cellular networks: Deployment, challenges, and applications, 2019.
- [33] Karl Hansson, Siril Yella, Mark Dougherty, and Hasan Fleyeh. Machine learning algorithms in heavy process manufacturing. *American Journal of Intelligent Systems*, 6(1):1–13, 2016.
- [34] Abbas Keramati, Hajar Ghaneei, and Seyed Mohammad Mirmohammadi. Developing a prediction model for customer churn from electronic banking services using data mining. *Financial Innovation*, 2(1):10, 2016.
- [35] Adrian Carrio, Carlos Sampedro, Alejandro Rodriguez-Ramos, and Pascual Campoy. A review of deep learning methods and applications for unmanned aerial vehicles. *Journal of Sensors*, 2017, 2017.
- [36] Or Sharir, Barak Peleg, and Yoav Shoham. The cost of training NLP models: A concise overview. *arXiv preprint arXiv:2004.08900*, 2020.

- [37] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- [38] Nathan Benaich and Ian Hogarth. State of AI report 2020. 2020. Available at www.stateof.ai
- [39] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. *arXiv preprint arXiv:1906.02243*, 2019.
- [40] Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020.
- [41] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.
- [42] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.
- [43] Diana Marculescu, Dimitrios Stamoulis, and Ermao Cai. Hardware-aware machine learning: Modeling and optimization. In *Proceedings of the International Conference on Computer-Aided Design, ICCAD ’18*, New York, NY, USA, 2018. Association for Computing Machinery.
- [44] Lucas Bernardi, Themistoklis Mavridis, and Pablo Estevez. 150 successful machine learning models: 6 lessons learned at Booking.com. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1743–1751, 2019.
- [45] Danilo Sato, Arif Wider, and Christoph Windheuser. Continuous delivery for machine learning, 2019. Available at <https://martinfowler.com/articles/cd4ml.html>
- [46] Ram Ananth, Seph Mard, and Peter Simon. Opening the “black box”: The path to deployment of AI models in banking, white paper. *DataRobot and REPLY AVANTAGE*, 2019.
- [47] Prudential Regulation Authority. Model risk management principles for stress testing, 2018.
- [48] ECB TRIM Guide. Guide for the targeted review of internal models (TRIM). *European Central Bank*, 2017.
- [49] Eric Breck, Marty Zinkevich, Neoklis Polyzotis, Steven Whang, and Sudip Roy. Data validation for machine learning. In *Proceedings of SysML*, 2019.
- [50] Yingnong Dang, Qingwei Lin, and Peng Huang. AIOps: real-world challenges and research innovations. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 4–5. IEEE, 2019.
- [51] Andrew Zhai, Hao-Yu Wu, Eric Tzeng, Dong Huk Park, and Charles Rosenberg. Learning a unified embedding for visual search at Pinterest, 2019.
- [52] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. In *Advances in neural information processing systems*, pages 2503–2511, 2015.
- [53] Janis Klaise, Arnaud Van Looveren, Clive Cox, Giovanni Vacanti, and Alexandru Coca. Monitoring and explainability of models in production. *arXiv preprint arXiv:2007.06299*, 2020.
- [54] Klaus Ackermann, Joe Walsh, Adolfo De Unánue, Hareem Naveed, Andrea Navarrete Rivera, Sun-Joo Lee, Jason Bennett, Michael Defoe, Crystal Cody, Lauren Haynes, et al. Deploying machine learning models for public policy: A framework. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 15–22, 2018.
- [55] Tom Diethe, Tom Borchert, Eno Thereska, Borja Balle, and Neil Lawrence. Continual learning in practice. *arXiv preprint arXiv:1903.05202*, 2019.
- [56] Syed Muslim Jameel, Manzoor Hashmani, Hitham Alhussian, Mobashar Rehman, and Arif Budiman. A critical review on adverse effects of concept drift over machine learning classification models. *International Journal of Advanced Computer Science and Applications*, 11, 01 2020.

- [57] Bilge Celik and Joaquin Vanschoren. Adaptation strategies for automated machine learning on evolving data. *arXiv preprint arXiv:2006.06480*, 2020.
- [58] Andrés R Masegosa, Ana M Martínez, Darío Ramos-López, Helge Langseth, Thomas D Nielsen, and Antonio Salmerón. Analyzing concept drift: A case study in the financial sector. *Intelligent Data Analysis*, 24(3):665–688, 2020.
- [59] Daniel Langenkämper, Robin van Kevelaer, Autun Purser, and Tim W Nattkemper. Gear-induced concept drift in marine images and its effect on deep learning classification. *Frontiers in Marine Science*, 7:506, 2020.
- [60] Jan Zenisek, Florian Holzinger, and Michael Affenzeller. Machine learning based concept drift detection for predictive maintenance. *Computers & Industrial Engineering*, 137:106031, 2019.
- [61] Jeffrey C Schlimmer and Richard H Granger. Incremental learning from noisy data. *Machine learning*, 1(3):317–354, 1986.
- [62] Arif Wider and Christian Deger. Getting smart: Applying continuous delivery to data science to drive car sales, 2017. Available at <https://www.thoughtworks.com/insights/blog/getting-smart-applying-continuous-delivery-data-s>
- [63] David Leslie. Understanding artificial intelligence ethics and safety. *arXiv preprint arXiv:1906.05684*, 2019.
- [64] Changhee Han, Leonardo Rundo, Kohei Murao, Takafumi Nemoto, and Hideki Nakayama. Bridging the gap between AI and Healthcare sides: towards developing clinically relevant AI-powered diagnosis systems, 2020.
- [65] John Mark Michael Rumbold and Barbara Pierscionek. The effect of the general data protection regulation on medical research. *Journal of medical Internet research*, 19(2):e47, 2017.
- [66] Syed Mohamed Aljunid, Samrit Srithamrongsawat, Wen Chen, Seung Jin Bae, Raoh-Fang Pwu, Shunya Ikeda, and Ling Xu. Health-care data collecting, sharing, and using in Thailand, China mainland, South Korea, Taiwan, Japan, and Malaysia. *Value in Health*, 15(1):S132–S138, 2012.
- [67] Mustafa Suleyman and Dominic King. The Information Commissioner, the Royal Free, and what we've learned, 2017.
- [68] Cathy O’Neil. *Weapons of math destruction: How big data increases inequality and threatens democracy*. Broadway Books, 2016.
- [69] Robert Soden, Dennis Wagenaar, Dave Luo, and Annegien Tijssen. Taking ethics, fairness, and bias seriously in machine learning for disaster risk management, 2019.
- [70] Solon Barocas, Moritz Hardt, and Arvind Narayanan. Fairness in machine learning. *NIPS Tutorial*, 1, 2017.
- [71] Nantheera Anantrasirichai and David Bull. Artificial intelligence in the creative industries: A review, 2020.
- [72] Yisroel Mirsky and Wenke Lee. The creation and detection of deepfakes: A survey. *arXiv preprint arXiv:2004.11138*, 2020.
- [73] Sathappan Muthiah, Patrick Butler, Rupinder Paul Khandpur, Parang Saraf, Nathan Self, Alla Rozovskaya, Liang Zhao, Jose Cadena, Chang-Tien Lu, Anil Vullikanti, et al. Embers at 4 years: Experiences operating an open source indicators forecasting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 205–214, 2016.
- [74] M-C Lai, M Brian, and M-F Mamzer. Perceptions of artificial intelligence in healthcare: findings from a qualitative survey study among actors in France. *Journal of Translational Medicine*, 18(1):1–13, 2020.
- [75] Ramprakash Ramamoorthy, P Satya Madhuri, and Malini Christina Raj. AI from labs to production - challenges and learnings. Santa Clara, CA, May 2019. USENIX Association.
- [76] Mark Sendak, Madeleine Clare Elish, Michael Gao, Joseph Futoma, William Ratliff, Marshall Nichols, Armando Bedoya, Suresh Balu, and Cara O’Brien. “The human body is a black box”: Supporting clinical decision-making with deep learning. In *Proceedings of the 2020*

- Conference on Fairness, Accountability, and Transparency*, FAT* ’20, page 99–109, New York, NY, USA, 2020. Association for Computing Machinery.
- [77] Information Commissioner’s Office. Project ExplAIn interim report, 2019. Available at <https://ico.org.uk/about-the-ico/research-and-reports/project-explain-interim-report/>.
 - [78] Ram Shankar Siva Kumar, David O’Brien, Kendra Albert, Salomé Viljöen, and Jeffrey Snover. Failure modes in machine learning systems. *arXiv preprint arXiv:1911.11034*, 2019.
 - [79] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, 2018.
 - [80] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.
 - [81] Ram Shankar Siva Kumar, Magnus Nystrom, John Lambert, Andrew Marshall, Mario Goertzel, Andi Comissoneru, Matt Swann, and Sharon Xia. Adversarial machine learning-industry perspectives. Available at SSRN 3532474, 2020.
 - [82] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 19–35. IEEE, 2018.
 - [83] Oscar Schwartz. In 2016 Microsoft’s racist chatbot revealed the dangers of online conversation. *IEEE Spectrum*, 11, 2019.
 - [84] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction APIs. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 601–618, 2016.
 - [85] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333, 2015.
 - [86] Michael Veale, Reuben Binns, and Lilian Edwards. Algorithms that remember: model inversion attacks and data protection law. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 376(2133):20180083, 2018.
 - [87] Chip Huyen. What I learned from looking at 200 machine learning tools, 2020. Available at <https://huyenchip.com/2020/06/22/mlops.html>.
 - [88] Kumar Venkateswar. Using Amazon SageMaker to operationalize machine learning. 2019.
 - [89] AML Team. AzureML: Anatomy of a machine learning service. In *Conference on Predictive APIs and Apps*, pages 1–13, 2016.
 - [90] Jeremy Hermann and Mike Del Balso. Meet Michelangelo: Uber’s machine learning platform. URL <https://eng.uber.com/michelangelo>, 2017.
 - [91] Denis Baylor, Kevin Haas, Konstantinos Katsiapis, Sammy Leong, Rose Liu, Clemens Menwald, Hui Miao, Neoklis Polyzotis, Mitchell Trott, and Martin Zinkevich. Continuous training for production ML in the TensorFlow extended (TFX) platform. In *2019 {USENIX} Conference on Operational Machine Learning (OpML 19)*, pages 51–53, 2019.
 - [92] Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, et al. Accelerating the machine learning lifecycle with mlflow. *IEEE Data Eng. Bull.*, 41(4):39–45, 2018.
 - [93] Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. Beyond accuracy: Behavioral testing of NLP models with CheckList. In *Association for Computational Linguistics (ACL)*, 2020.
 - [94] Stephen H Bach, Daniel Rodriguez, Yintao Liu, Chong Luo, Haidong Shao, Cassandra Xia, Souvik Sen, Alex Ratner, Braden Hancock, Houman Alborzi, et al. Snorkel DryBell: A case study in deploying weak supervision at industrial scale. In *Proceedings of the 2019 International Conference on Management of Data*, pages 362–375, 2019.
 - [95] Paroma Varma and Christopher Ré. Snuba: Automating weak supervision to label training data. In *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, volume 12, page 223. NIH Public Access, 2018.
 - [96] Curtis G. Northcutt, Lu Jiang, and Isaac L. Chuang. Confident learning: Estimating uncertainty in dataset labels, 2019.

- [97] Neil Lawrence. Modern data oriented programming, 2019. Available at <http://inverseprobability.com/talks/notes/modern-data-oriented-programming.html>
- [98] Tom Borchert. Milan: An evolution of data-oriented programming, 2020. Available at <https://tborchertblog.wordpress.com/2020/02/13/28/>
- [99] Alexander Lavin and Gregory Renard. Technology readiness levels for machine learning systems. *arXiv preprint arXiv:2006.12497*, 2020.
- [100] Martin Zinkevich. Rules of machine learning: Best practices for ML engineering. *URL: https://developers.google.com/machine-learning/guides/rules-of-ml*, 2017.



Data management for production quality deep learning models: Challenges and solutions[☆]



Aiswarya Raj Munappy ^{a,*}, Jan Bosch ^a, Helena Holmström Olsson ^b, Anders Arpteg ^c,
Björn Brinne ^c

^a Department of Computer Science and Engineering, Chalmers University of Technology, Hörselgången 11, 412 96, Gothenburg, Sweden

^b Department of Computer Science and Media Technology, Malmö University, Nordenskiöldsgatan 1, 205 06, Malmö, Sweden

^c Peltarion - the operational AI platform, Holländargatan 17, 111 60, Stockholm, Sweden

ARTICLE INFO

Article history:

Received 12 October 2021

Received in revised form 22 March 2022

Accepted 3 May 2022

Available online 12 May 2022

Keywords:

Deep learning
Data management
Production quality DL models
Challenges
Solutions
Validation

ABSTRACT

Deep learning (DL) based software systems are difficult to develop and maintain in industrial settings due to several challenges. Data management is one of the most prominent challenges which complicates DL in industrial deployments. DL models are data-hungry and require high-quality data. Therefore, the volume, variety, velocity, and quality of data cannot be compromised. This study aims to explore the data management challenges encountered by practitioners developing systems with DL components, identify the potential solutions from the literature and validate the solutions through a multiple case study. We identified 20 data management challenges experienced by DL practitioners through a multiple interpretive case study. Further, we identified 48 articles through a systematic literature review that discuss the solutions for the data management challenges. With the second round of multiple case study, we show that many of these solutions have limitations and are not used in practice due to a combination of four factors: high cost, lack of skill-set and infrastructure, inability to solve the problem completely, and incompatibility with certain DL use cases. Thus, data management for data-intensive DL models in production is complicated. Although the DL technology has achieved very promising results, there is still a significant need for further research in the field of data management to build high-quality datasets and streams that can be used for building production-ready DL systems. Furthermore, we have classified the data management challenges into four categories based on the availability of the solutions.

© 2022 The Authors. Published by Elsevier Inc. This is an open access article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>).

1. Introduction

Deep learning (DL) is fundamentally a neural network with three or more layers. The software systems that employ DL can learn multiple levels of representations (Zhang et al., 2019). DL is a subset of machine learning techniques that use supervised and/or unsupervised strategies to automatically learn hierarchical representations in deep architectures for classification (Bengio, 2000; Ranzato et al., 2007) contrary to the conventional learning methods, which use shallow-structured learning architectures. Consequently, deep learning is now used extensively for image processing in healthcare applications (Litjens et al., 2017), Self Driving Cars (Daily et al., 2017; Rao and Frtuník, 2017),

2018), Virtual Assistants (Kepuska and Bohouta, 2018), Automatic Machine Translation (Bahdanau et al., 2014) and Fraud Detection (Roy et al., 2018). Large companies such as Facebook (Abadi et al., 2016), Google (Beaufays, 2015), Uber (Sergeev and Del Balso, 2018; Gruener et al., 2018), Amazon (Rastogi, 2018), Microsoft (Deng et al., 2014) have employed DL in a wide range of their products. For instance, Facebook uses DL to identify excessively promotional posts, spam, or clickbait (Abadi et al., 2016). Google incorporates DL into the search engine (Beaufays, 2015) to understand spoken commands and questions. Uber implements automated DL transcription technology to enable scalable, reliable, and quick validation of driver identity when drivers go online (Sergeev and Del Balso, 2018; Gruener et al., 2018). Amazon uses DL to improve customers' experience with Alexa skills (Rastogi, 2018). DL is one of the most refined machine learning techniques that does not often require feature engineering. However, DL is not a widely used technique among industries due to poor explainability, poor traceability, data dependencies, dataset incompleteness, and data management problems (Rao and Frtuník, 2018; Kahng et al., 2017).

[☆] Editor: Burak Turhan.

* Corresponding author.

E-mail addresses: aiswarya@chalmers.se (A.R. Munappy), jan.bosch@chalmers.se (J. Bosch), helena.holmstrom.olsson@mau.se (H.H. Olsson), anders.arpteg@gmail.com (A. Arpteg), bjorn@peltarion.se (B. Brinne).

Although DL models demand less domain expertise and are capable of learning automatically from input data (i.e. the features are not given by a human) (Zhang et al., 2016), understanding and explaining the learned models is extremely difficult. Even the most well-studied models that operate so well remain a mystery, requiring additional research. Consequently, changing the model, its network structure, and hyperparameters demands more effort and time. Further, it has inherent drawbacks, such as high sensitivity to underlying data compared to machine learning models (LeCun et al., 2015). Furthermore, data is treated as a first-class citizen, equal to code, and therefore data management has a significant impact on model accuracy. Although data is everywhere, searching for the right ones in the right quantity itself becomes a challenge (Whang and Lee, 2020). Moreover, the collected data should be clean and validated to rectify problems in the data, so that it will not affect the performance of the DL model (Whang and Lee, 2020). Even after collecting the right data and cleaning it, data quality may still be an issue during model training (Whang and Lee, 2020). The evolving nature of data leads to problems after the model deployment as well (Munappy et al., 2019). Thus, data management becomes a challenge that affects all phases of a DL model development pipeline.

The objective of this paper is to explore, analyze, and understand the data management challenges encountered by industry practitioners when developing and maintaining DL systems. Further, this study aims to analyze why data management for DL cannot be solved by existing solutions from other domains such as Big Data, data analytics, machine learning, and data mining. Using a multiple interpretive case study, we identified the data management challenges for production-quality DL models and mapped them with the corresponding data lifecycle phase, which is published as a conference paper (Munappy et al., 2019). This study is an extension of it by incorporating the potential solutions to data management challenges from the previous research using a systematic literature review (Kitchenham, 2004). Furthermore, we validated the solutions in the second round of multi-case study research with companies that work on real-world DL projects.

The contribution of this paper is four-fold. First, we identify the data lifecycle phases and describe the data management challenges for DL at each phase of the data lifecycle through a multiple case study. Second, we present the solutions that can mitigate the data management challenges discussed in previous research through a systematic literature review. Third, we analyze why not all of these identified solutions presented in the literature are applicable in practice. Based on the identified limitations, we classify the challenges into four categories. Finally, we identify open research challenges in data management for DL and present them as future research directions.

The remainder of this paper is organized as follows. Background is presented next in Section 2 followed by the description of research method in Section 3. Section 4 presents an overview of the data lifecycle phases and data management challenges encountered at each phase. Section 5 presents the solutions for data management challenges. Section 6 shows the categorization of challenges according to the availability of solutions. The findings and research implications of the study are discussed and concluded in Section 7.

2. Background

Over the past few years, DL has spawned a tremendous collection of ideas and techniques that were previously believed to be infeasible. At first glance, this collection of ideas appears to be incoherent and dissimilar. However, over time, patterns and approaches evolve, and today DL (LeCun et al., 2015) represents a

significant step forward in overcoming the challenges. DL became the center of attraction after Krizhevsky et al. (2012) corroborated the significant performance of a Convolutional Neural Network (CNN) (Krizhevsky et al., 2012) based model on a challenging large-scale visual recognition task (LeCun et al., 1989) in 2012.

2.1. Data - The fuel for DL models

Digital data, in all its forms and sizes, is exploding at an incredible rate (National Security Agency, 2013). It also results in a significant paradigm shift in modern scientific research, with data-driven discovery becoming the norm (Manyika et al., 2011). Big data presents unprecedented challenges to harnessing data and information because of the sheer volume of data available today. On the one hand, it presents big opportunities and transformative potential for various sectors. Deep neural networks are trained using massive datasets to imitate human intelligence. Through a hierarchical learning process, DL algorithms extract high-level, complex abstractions as data representations. At each level of the hierarchy, complex abstractions are learned based on comparatively smaller abstractions formulated at the previous level. There are multiple high-performance algorithms in DL that are designed for diverse purposes. No algorithm, however, can guarantee the same results across all datasets, which is a clear indication of the importance of data and its impact on the output of DL models. The core benefit of DL is the analysis and learning of tremendous amounts of unsupervised data, which makes it a useful tool for Big Data Analytics even when raw data is mostly unlabeled and uncategorized. Furthermore, DL algorithms belong to the representation learning class, which has the capability of handling raw data and automatically extracting useful features as the representations (LeCun et al., 2015). As a result, data quality is crucial in determining the performance of a DL model. This combined power of Big Data and Deep Learning when they are working together clearly illustrates the considerable synergy between them.

2.2. Synergy between big data and DL

Machine learning and deep learning algorithms are capable of extracting every last detail from the input data, which is then utilized to develop new rules to fulfill the function (Labrinidis and Jagadish, 2012). Data and DL forms a symbiotic relationship in which DL is useless without data and data management is almost impossible to overcome without DL. Business decisions, previously reliant on speculation or painstakingly crafted models of reality, are now based on big data (Oguntimilehin and Ademola, 2014). The sheer volume and variety of data ingested by modern analytical pipelines considerably enhances the links between data integration and machine learning (Dong and Rekatsinas, 2018). Data management systems are increasingly using AI models like machine learning to automate parts of data life cycle tasks. Examples include data cataloging and inferring the schema of raw data (Halevy et al., 2016). Data analytics drives almost every prospect of our contemporary society, including mobile services, retail manufacturing, financial services, life sciences, and physical sciences (Oguntimilehin and Ademola, 2014). Established companies and newcomers alike prefer to use data-driven tactics to develop, compete, and gain value from deep and up-to-date information in most industries (Manyika et al., 2011). However, in the current scenario organizations struggle with collecting, integrating, and managing the data. Instead of solving these data issues, DL will only make them more noticeable.

2.3. Data management for DL models

The dimensions of big data are marked by three Vs namely Volume, Velocity, and Variety. Volume denotes the amount of data, variety denotes the number of types of data and velocity denotes the speed of data processing (Tole et al., 2013). The expansion of all three qualities results in the issues of big data management. As users come up with new ways to scrub and process data, the amount of data that can be extracted from the digital universe continues to grow.

Data management for DL can be defined as a process that includes collecting, processing, analyzing, validating, storing, protecting, and monitoring data to ensure the consistency, accuracy, and reliability of the data. Industry products that make use of tremendous volume of digital data can successfully employ DL. However, real-world data needs to be processed and managed before fed as input to the DL models. Training a DL model with such massive and variegated data sets is challenging, and several aspects need to be considered. e.g. data sparsity, redundancy, and missing values. To ensure the high performance from DL models, a set of good data management practices such as data pipelines (Raj et al., 2020) and DataOps (Munappy et al., 2020) should be followed from data collection, through data processing and analysis, dataset preparation, and deployment of the model. Wang et al. (2016) describe how certain challenges like data dependency, memory management, concurrency, data inconsistency can be solved by combining database techniques and deep neural networks. Moreover, ML based data management solutions have a GUI to help understand, visualize, and curate data for training, uncover corrupted data like mislabeled examples, and identify difficult edge cases. Solutions tie into an ML model and its data to determine the data that is helpful, hurtful, or useless for training algorithms. However, they are not good at multiple adjacencies like data labeling, label split balancing, embeddings-as-a-service, model monitoring, and model robustness verification.

Popular companies like Apple (Chen and Lin, 2014), Facebook, Microsoft is collecting a copious amount of data daily through applications like Siri, Google translator, Bing voice search (Jones, 2014) to provide a variety of other services such as reminders, weather reports, personalized recommendations. Although big data can render numerous opportunities, it also enforces consequential engineering challenges (Najafabadi et al., 2015). X. W. Chen et al. describe the big data challenges such as streaming data, high-dimensional data, scalability of models, and distributed computing (Tur and De Mori, 2011). However, in these papers, DL is considered a solution for the management of data. Data management challenges involved in implementing DL models are not seriously considered, and our paper intends to focus on that perspective.

3. Research methodology

We have used a three-step research process as shown in Fig. 1. In the first step, we identified data management challenges using interpretative multi-case research. As the second step, we conducted a systematic literature review and identified potential solutions for the identified challenges. Finally, and as the third step, we conducted a second round of multiple case study to validate the applicability of identified solutions on real-world industrial datasets. Detailed steps of the research process are illustrated in Fig. 2

3.1. Definition of research questions

The primary objective of this study is to identify data management challenges and solutions specific to DL in real-world settings. The secondary objective is to identify the open research questions in the area of data management for DL. We developed the following research questions to achieve our high-level goal:

- RQ1. What are the data management challenges experienced in the industry while developing DL models?**
- RQ2. What solutions are proposed in literature to address the identified challenges?**
- RQ3. What are the limitations of the existing solutions, and what remain as open challenges in data management for DL?**

3.2. Step 1: Interpretive multiple-case study (Exploration)

A multiple case study research method was adopted in this study. According to Yin et al. a case study is most suitable for 'how' and 'why' questions, as well as exploratory 'what' questions (Yin, 2013). In this study, 'what' questions are the key research questions justifying the choice of a case study approach. Further, Stuart et al. (2002) and Meredith (1998) propose that a case study can be considered as the appropriate method to explore new phenomena and generate new knowledge. Furthermore, a multiple case study method facilitates the exploration of the real-life challenges in its context through a variety of lenses (Baxter et al., 2008; Yin, 2003) and enhances the robustness of research findings, compared to a single case study, by reducing the risk of observer bias (Eisenhardt, 1989). Although our primary source of data collection is interviews, information collected through ethnographical observations and minutes from meetings are also incorporated by the two authors from the company wherever necessary, which in turn implements triangulation. The objective of this study is to identify challenges specifically concerned to the management of data in different real-world DL applications. An interpretive multiple-case study approach that adheres to the guidelines by Runeson et al. (Runeson and Höst, 2009) and Verner et al. (Verner et al., 2009) is employed in this research. The challenges identified are grounded on our interpretations of the experiences of experts who build and maintain DL systems in a real-time scenario with real-world datasets. The overall research design and the major steps in the research process of the study are described below.

3.2.1. Overview of Deep Learning use cases

This section describes real-world DL cases that has been chosen for this research. We focus on the system including the DL model, even if our focus is predominantly on the DL part of the system. DL use cases were selected based on the availability of experts working on it. The DL systems discussed are *Online recommender service*, *Medical imaging*, *Energy Prediction*, *Real-Estate Forecast*, *Manufacturing*, and *Financial Systems* as shown in Table 1. All of these are using real-world dataset and are operational.

Case A – Online recommender services: Case A is an online recommender system used by an electric vendor. DL components in recommender systems are trained on user reviews and the purchase history. When a customer visits the website, the recommender system predicts users' interest and recommends electric products based on previous customer reviews and purchase history. Online recommender services help the company boost sales by leveraging the power of data. Many customers tend to look at the website for their recommendations. Personalized recommendations from the system thus increase customer satisfaction and thus customer retention. It not only creates personalized



Fig. 1. Three-step Research Process.

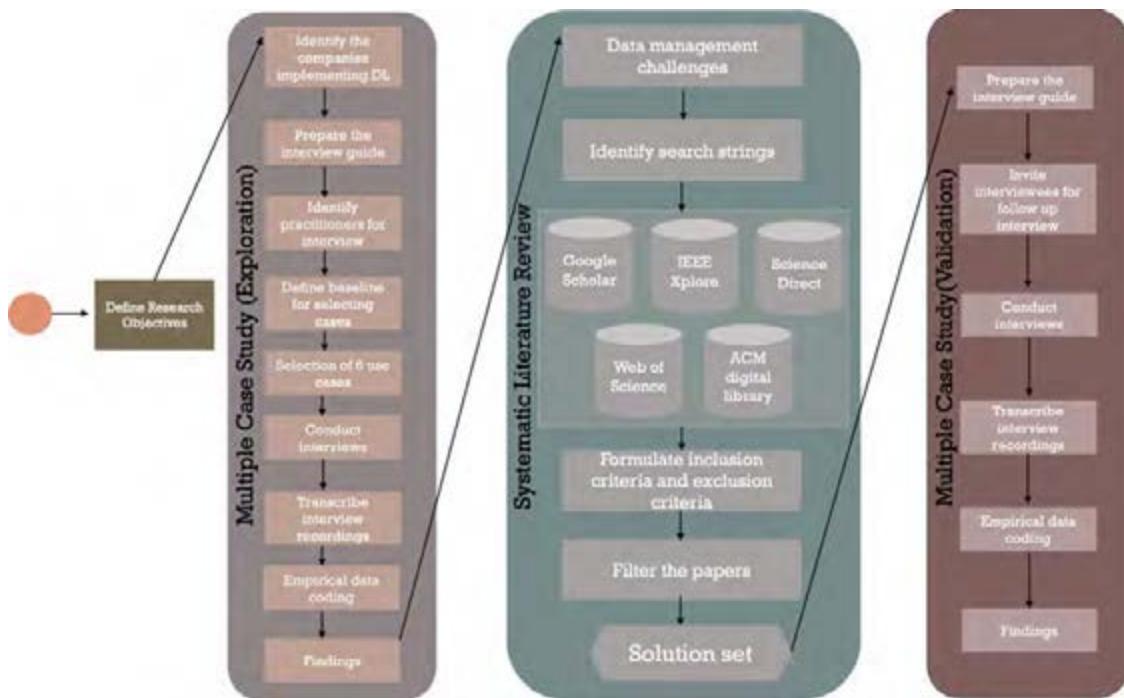


Fig. 2. Research methods and process for conducting the study.

Table 1
Deep Learning use cases and description.

Case No	Deep Learning Use case	Description
Case A	Online Recommender Service	Predicts users' interest and recommends electric products for them based on previous customer reviews and purchase history
Case B	Medical imaging	Automated classification of skin lesions into malicious and benign
Case C	Energy Prediction	Wind power is predicted based on the meteorological data
Case D	Real-Estate Forecast	Predicts the property prices based on historical data
Case E	Manufacturing	Predicts the quality of cartons made from pulp
Case F	Financial Systems	Automated classification of transactions into fraudulent and normal

informational flows independently for each user, but also takes into account the behavior of all users of a service. Along with the information about users' interactions with items, there is usually data describing users and items separately. This data could be assorted and heterogeneous – items and users contain textual descriptions, numerical characteristics, categorical features, images, and other types of data.

Case B – Medical imaging: Case B is a melanoma detection system. Melanoma is a type of skin cancer, which is not usual like basal cell and squamous carcinoma, but it has dangerous implications since it tends to migrate to other parts of the body. Therefore, early detection is required to prevent it from spreading to other parts; otherwise, it becomes incurable. The skin cancer

detector not only intends to diagnose whether a person has skin cancer or not, but also the type of cancer and severity. Here, the DL system is used for the diagnostic classification of dermoscopic images of lesions of melanocytic origin. Although datasets such as MED-NODE, ISIC Archive, are publicly available, dealing with real-world data is still challenging. Automated classification of skin lesions using images is a difficult task because of the unavailability of fine-grained varieties of the appearance of skin lesions. In this use case, datasets are formed over several years by working in close collaboration with clinics. The company has regulations on the usage of the dataset and the data is not allowed to leave the servers. With these regulations, the practitioners conform to the rules specific to the dataset and move the code and model to

the server where data is stored for developing the DL system. The melanoma detector is still not production-ready.

Case C – Wind power prediction: Case C is a wind power predictor. Wind power is weather-dependent, and therefore it is irregular and fluctuates over different time scales. Thus, accurate forecasting of wind power is considered as a major contribution to reliable large-scale wind power integration. Here, the DL system is utilized to predict accurately the amount of electricity and power the wind turbines are going to produce within 24 to 48 h so that an accurate report can be furnished to the power companies for which energy is supplied. In this use case, a combination of wind and weather is predicted, from which the power generated by the wind turbines is calculated. The wind power is predicted based on the meteorological data obtained from the National meteorological agency. Deep Learning is used to forecast weather and thereby predicting the wind power that can be generated in the future. The power companies have strict requirements in terms of the amount of power they are going to deliver, and penalties should be paid if they cannot deliver the reported energy.

Case D – Real-Estate Forecast: Case D is a system used for real-estate forecasting. Predicting property values is of significant interest to various parties in an economy. Estimation of the house price is important to prospective homeowners, developers, investors, appraisers, tax assessors, and other real estates market participants, such as mortgage lenders and insurers. Real-estate investors and portfolio managers prepare and conduct their investment decisions grounded on periodic evaluations of their real-estate portfolios. Individuals are interested in knowing the values of their properties before determining their list prices. Tax authorities levy property taxes based on estimates of the value of the properties. Banks and mortgage providers conduct housing collateral valuations to qualify the borrowers for their mortgage applications. Initially, house price was predicted based on the comparison between cost and sale price and there were neither accepted standards nor certification processes. Therefore, the house price prediction system was used to fill up the existed information gap, and it also enhanced the efficiency of the real estate market. The house price prediction case was initially built on a traditional assorted database system where SQL queries and data pipeline scripts were used, whereas now it utilizes DL techniques where the model is trained with historical sales data about properties, geography, and demography in the Swedish market. The house price prediction system is a long-running DL system deployed in production and is used by many banks in Sweden.

Case E – Manufacturing: Case E is from the manufacturing domain, which is a paper mill industry that produces paper from pulp. The pulp is dried to form cartons and cardboard, which is further used for making milk cartons. The company produces a huge amount of paper board every year and wants to keep material costs as low as possible while retaining high quality. Paper mills gather large amounts of data, which provides them with ever-growing visibility into their processes, due to the rapidly increasing availability of instrumentation and the usage of centralized data historians. The quality of the paper board is predicted based on data from process sensors and images of wood fibers taken with the PulpEye technology. A DL component is incorporated in the system to predict the quality of the resulting product based on all the measurements in the machine and measurements on the pulp that goes in. Further, there are also images of what happens at the start of the machine, and microscope images of the fibers in the pulp. The DL system serve as a foundation to control the manufacturing process so that the

Table 2
Description of Use cases and Roles of the interviewees.

Case	Use case	Interviewed Experts	
		ID	Role
A	Recommending products to the users in a personalized fashion	P1	Principal Data Scientist
B	Predicting the wind power using the historical weather data	P2 P3 P4 P5	Data Scientist Head of Data Analytics team Data Scientist AI Research Engineer
C	Estimating and predicting the price of houses	P2 P3	Data Scientist Head of Data Analytics team
D	Automated classification of skin lesions into benign and malignant	P2 P3 P4 P5	Data Scientist Head of Data Analytics team Data Scientist AI Research Engineer
D	Detecting the credit card frauds during gaming	P2 P3	Data Scientist Head of Data Analytics team
E	Predicting quality of paper boards	P4 P5	Data Scientist AI Research Engineer

same quality could be maintained with less input material and waste.

Case F – Financial Systems: Case F is a financial fraud detection system. Frauds in finance still amount to significant loss of money. Hackers and fraudsters all over the world are experimenting with new techniques to perpetrate financial fraud. Therefore, trusting financial fraud detection systems programmed based on the conventional rule-based method alone will not serve the purpose. This is where Deep Learning shines as a unique solution. The DL system utilizes customer details such as payment history, activity history and payment request data such as payment method, amount, location, etc. For fraud detection, post-payment signals of anomalous pay are also considered. In the current world, fraud detection requires a complete strategy that matches data points with activities to determine what is wrong. When it comes to modeling fraud detection as a classification problem, the main challenge is that in the real world, the majority of the transactions are not fraudulent. However, to train DL systems, counterexamples are also required.

Based on the study with the aforementioned use cases, data management challenges are identified. This study presents a set of clearly explained data management challenges faced by practitioners while integrating DL components in real-world software systems. We have also classified the challenges according to the data life cycle phase in which they are encountered.

3.2.2. Expert interviews

The primary objective of our study was to explore data management challenges encountered while implementing DL systems in real-world settings. Each case in the study pertains to a software-intensive system that incorporates DL components developed at an organization. For the study, a sample pool of DL experts who has an experience of minimum 3 years working exclusively on DL systems and works in seven different domains were selected by their expertise in the area of study. The selected seven practitioners include two authors of this paper. From the acknowledgment in the literature (and our experiences when soliciting interviewees), it can be inferred that only a few experienced practitioners are skilled in the area of intersection between DL and SE. Table 2 illustrates the roles of our interviewees in implementing use cases across multiple domains.

3.2.3. Data collection

Semi-structured interviews were used to acquire qualitative data. Based on the objective of the research to explore data management challenges for DL systems, an interview guide with 40 questions categorized into four sections was formulated. The first and second sections focused on the background of the interviewee. The third section concentrated on the importance of data in various projects and the last section inquired in detail about data management, the challenges faced during every phase of the data processing pipeline. The interview guide was reviewed by the authors and some additional questions were added, a few similar questions were merged, and some less relevant questions were removed, finally forming an interview protocol with 36 questions spread across five different categories. All interviews were face-to-face except for one which was done via video conference and each interview lasted 45 to 55 min. All the interviews were recorded with the permission of respondents and were transcribed later for analysis.

3.2.4. Data analysis

The audio recordings were sent to be transcribed after the interviews, and the first author wrote a synopsis of each interview, summarizing the important points of discussion. After transcription, the analytical insights from the summary were cross-checked repeatedly with the audio recordings and interview transcripts. The results of data analysis were checked by second and third author of this paper to reduce the bias. A theoretical thematic data analysis approach was opted for coding (Maguire and Delahunt, 2017). First, the author coded each segment of the interview transcript that was relevant to or captured something interesting about data in NVivo. In the first iteration, the aim was to identify the phases of the data life cycle. After identifying the phases as shown in Fig. 2, a second iteration was performed to code the data management challenges encountered in each phase by setting high-level themes as (i) *Data Collection*, (ii) *Data Exploration*, (iii) *Data Preprocessing*, (iv) *Dataset Preparation*, (v) *Data Testing*, (vi) *Deployment*, (vii) *Post-deployment*. Further, the codes such as problem description, implications, empirical basis, examples etc. were formed. The results deduced from the analysis were tabulated and sent to the authors for comments, and then the final summary of the cases and mapping were sent to the interviewees for validating the inferred results.

3.3. Step 2: Systematic literature review

As the second step in our research, we conducted a systematic literature review (SLR) based on the guidelines proposed by Kitchenham (2004). In this study, the goal of the SLR was to identify the solutions for data management challenges identified through multiple case study. As we already identified the data management challenges through interview study, we have only searched for solutions to the identified challenges in the literature. We have not attempted to expand the list of data management challenges through literature review. According to Kitchenham, systematic literature reviews not only aggregate all existing evidence on a research question, but also intend to support the development of evidence-based guidelines for practitioners. Further, SLRs help to identify less explored areas and thus provide a framework to position new research activities. Moreover, systematic literature reviews aim to identify, analyze, and interpret all relevant studies on the topic of interest (Kitchenham, 2004). In this study, our topic of interest was data management challenges for DL and solutions. From the topic, the aim was to investigate the data management challenges experienced by practitioners while implementing DL components in software-intensive systems. The interview study was performed followed

by a literature review to identify the data management challenges. Further, we identified proposed solutions from literature as well as from the practitioners. The SLR process is summarized as three phases: (1) planning the review, (2) conducting the review, and (3) reporting the review. Definition of research questions, identification of research, selection of primary studies, study quality assessment, data extraction, data analysis, and synthesis are the steps that are described in detail below.

3.3.1. Selection of relevant studies

To analyze all available empirical materials specific to the objective of this research, we started with the formulation of a formal search strategy after defining our research goals and questions.

Search strategy. Google Scholar, IEEE Xplore, Web of Science, and ACM Digital Library databases were searched since they include journals and conferences focusing on data management as well as DL. These databases allow us to perform keyword searches. The search was conducted in April 2020 and therefore this SLR includes studies that were published before the date. To collect the maximum number of relevant papers, we did not restrict ourselves to selected journals/conferences.

Search string. We formulated search strings that included searched the three important keywords in our four research questions. Further, we supplemented the keywords with their synonyms, resulting in the following search string: (((Data) AND (metadata OR labeling OR aggregation OR GDPR OR duplication OR redundant OR heterogeneous OR dirty OR categorical OR transformation OR sequences OR time-series OR extraction OR overfitting OR regularization OR cross-validation OR feedback) AND (challenges OR problems OR issues OR characteristics) OR (technique OR methods OR approaches)) AND (data OR data management OR data analytics OR machine learning OR data mining)) AND ("(industry OR company OR validation OR empirical)")

3.3.2. Study selection

We formulated the inclusion criteria and exclusion criteria to select the papers relevant to the study.

Inclusion criteria (IC)

1. A research paper that describes the data management for deep learning/machine learning in industrial settings,
2. A research paper that explicitly describes solution for a particular identified data management challenge with validation

Exclusion criteria (EC)

1. Non-scientific papers
2. Non-English and duplicates
3. Proposals without industrial/academic validation

We conducted a literature review to identify the data management methods for solving challenges associated with the data for DL. DL is a widely used technique among large-scale companies like Facebook, Google, Uber, etc. We were only interested in identifying the papers that discuss data management solutions for DL models. As data and challenges around it are discussed in various contexts, solutions for data management challenges can be adopted from other data domains as well. Therefore, we extended our search to data analytics, data management, and data mining papers to identify solutions for data management challenges. We have only included the papers that validate the solution with some datasets.

The paper titles were examined to filter studies that were unrelated to our search objective. We reviewed the abstracts and keywords in the remaining studies to select relevant studies. In

many cases, an abstract and keywords were insufficient to establish whether a study was relevant. In such cases, we also went over the conclusions. Further, we filtered the remaining studies by applying the inclusion/exclusion criteria. The initial primary studies for the SLR are chosen from this final step. A complete list of included papers (primary studies as well as supporting studies) is provided in the Figshare.

3.3.3. Data extraction

We defined a data extraction process according to the guidelines provided in Kitchenham (2004) to identify relevant information from the 32 included primary studies that pertain to our research questions. Our data extraction process includes the following: First, we set up an Excel table to record ideas, concepts, and findings of each of the 58 papers.

3.3.4. Data analysis and synthesis

The solutions are classified according to the data life cycle phase in which they can be used to mitigate the challenge. The solution was analyzed to check if it was applicable in the context of deep learning. Furthermore, the solutions were again classified into expensive and cheap in terms of infrastructure and cost-effectiveness. Each of the identified solutions was thoroughly analyzed to interpret the reason for it being counted as a challenge in the deep learning context even with the solutions from other domains.

3.4. Step3: Interpretive multiple case study (Validation)

We have conducted a second round of study for validating the results obtained after performing the first two steps. The objective of this follow-up study is to validate the solutions we identified from the literature. We followed exactly the same procedure as step 1. The interviewees who participated in the first round of multiple case study were contacted through email. We shared the results of the first study in the form of a conference paper and the solutions from literature as a separate document, and requested them to go through those documents if they are interested in participating in the validation study. After 1.5 weeks, they were contacted again with an interview guide for validation study. Most of them showed interest in participating in the study. Therefore, we conducted the follow-up study with the interview guide we prepared for validating the solutions. All the interviews lasted for around 30–45 min. Interviewees were asked for the changes in the availability of solutions for the identified set of challenges. We also asked if they are familiar with the solutions from literature. Further, we inquired in detail about the solutions identified from literature and their reasoning for not using it in the industries. The interviews were recorded with the permission of interviewees and transcribed for performing data analysis. First author performed open coding and identified high level codes such as working solutions, used solutions, non-general solutions, partial solutions, reasons etc. The codes were reviewed by the second and third author.

3.5. Threats to validity

The interview study can have some threats to validity such as construct, internal, and external threats, credibility and transferability. This section explains the mitigation strategies used to administer these threats.

3.5.1. Construct validity

A few cases were removed from the results to verify construct validity, as some interviewers did not have a thorough comprehension of the concepts covered. Our study includes certain limitations with multiple interviews as a result of the screening process. This limitation, on the other hand, can be viewed as an opportunity for future research. The results of the interviews were disclosed to the participants in order to reduce researcher bias. We also created a semi-structured interview guide and delivered it to the interviewees prior to the interviews. Before the interview, the interviewees were supplied a brief synopsis of the topic to be discussed. We rephrased the question of whenever the response becomes off-topic, or asked them to elaborate when we received ambiguous answers. Further, confusions or lack of clarity were resolved by contacting the participants.

3.5.2. Internal validity

As the researcher only had limited access to the descriptions of the strategies, it was not possible to investigate about the other influential factors that can affect the final results. To minimize internal validity threats, two of the co-authors, who has in-depth knowledge about the data processed in the company, were asked to validate the findings. Moreover, the paper was sent to the steering committee for review before the final submission.

3.5.3. External validity

The presented study is derived from the cases studied with different teams in the domains of manufacturing, banking, business and healthcare. Some parts of the work can be seen in parts of the company differently. All company terminologies are normalized, and implementation details are given at the appropriate degree of abstraction (Bickman and Rog, 2008). We do not claim that the opportunities and challenges will be the same for companies from different disciplines.

3.5.4. Descriptive validity

Descriptive Validity refers to the accuracy and objectivity of the data collected during the case study. In order to mitigate the problems with factual accuracy and objectivity, all the interviews were recorded with the permission of interviewees. Further, interviewees were contacted through email whenever lack of clarity is encountered while transcribing the recordings. Two co-authors from the company also helped with certain company specific terminologies. Thus, we rule out the chance of misinterpreting what participants say and do, as well as their perspective on what is going on, which is a crucial way of detecting biases and misconceptions of what is interpreted.

3.5.5. Credibility

To increase the credibility of the results obtained through literature review, we conducted a second round of interview study with the practitioners participated in the first round.

3.5.6. Transferability

The degree to which qualitative research findings can be generalized or transferred to different contexts or settings is referred to as transferability. Due to the non-disclosure agreement with the participants of the case study, we have limitations in disclosing the entire details gathered during data collection. We agree that this impacts the transferability and to reduce this, we have tried to include details wherever possible with the help of anonymization.

A strength of our study is the use of both multi-case study approach and systematic literature review for framing the results. However, several factors present some potential threat to the validity of the systematic literature review.

**Fig. 3.** Data lifecycle phases.

Table 3
Mapping between data management challenges and DL use cases.

Phase	Challenge	Use cases of DL components					
		RS ^a	WPP ^b	HPP ^c	MD ^d	FFD ^e	MS ^f
Data Collection	Lack of labeled data	X	X	X	X	X	X
	Data Granularity	X	X				
	Shortage of diverse samples		X	X	X	X	
	Need for sharing and tracking techniques	X	X	X	X	X	
	Data Storage complying to GDPR	X					
Data Exploration	Statistical Understanding		X			X	X
	Deduplication Complexity	X	X	X	X	X	X
	Heterogeneity in data	X	X	X	X	X	
Data Preprocessing	Dirty data	X	X	X	X	X	X
	Managing sequences in data		X	X		X	
	Managing categorical data			X		X	
Dataset Preparation	Data Dependency	X	X	X	X	X	X
	Data Quality	X	X	X	X	X	X
Data Testing	Tooling	X	X	X	X	X	X
	Expensive Testing	X			X		X
Deployment	Data Extraction Methods	X	X	X	X	X	X
	Overfitting				X	X	
Post Deployment	Data sources and Distribution	X	X	X			
	Data drifts	X	X	X			
	Feedback loops	X					

^aRecommender System.^bWind Power Prediction.^cHouse Price Prediction.^dMelanoma Detection.^eFinancial Fraud Detection.^fManufacturing Systems.

3.5.7. Identification and selection of papers

Search string based approach was adopted for identifying and selecting the primary set of papers. This might have resulted in missing of relevant studies and in turn the solutions to the challenges. To mitigate this threat, we conducted a second round of interview study to validate the solutions and sought the help of experts to see if we missed any relevant solutions. Further, data collection period will have an impact on our findings. Many new papers would have been published after we did our search in various databases.

3.5.8. Exclusion of gray literature

Our study is also limited by the fact that we have not used gray literature or non-academic literature. According to Garousi et al. multi-vocal literature review with both academic and non-academic literature is used when there is limited studies on a specific topic. Since, deep learning is a mature field, we had 58 papers included in the study. Therefore, we did not incorporate the results from gray literature.

4. Data management challenges for DL

This section presents the findings from the first step of the three-step research process in Fig. 2 which is an exploratory interpretive multi-case study. First, we illustrate the phases of the data lifecycle. Then, we map the data management challenges to the corresponding phase as shown in Table 3 and describe the challenges. Data Collection, Data Exploration, Data Preprocessing, Dataset Preparation, Data Testing, Deployment, and Post-deployment are the 7 data lifecycle stages as shown in Fig. 3.

4.1. Data collection

The data lifecycle starts with the data collection phase. Data collection is the process of acquiring data from a wide range of sources that produces data. Lack of labeled data, data granularity, shortage of diverse samples, data sharing and tracking methods, data storage complying with GDPR are the challenges encountered during the phase of data collection.

4.1.1. Lack of labeled data

Description: Success of supervised DL algorithms is underpinned by labeled data with sufficient quality data labels that are required for training, testing, and validation. Data labels are obtained through the process of transcribing, tagging, and labeling significant features within the data. With high-quality, human-powered data labeling, companies can build and improve DL implementations. As a result of the disparate origins, unlabeled data and noisy labels increase the complexity. Further, to label data, metadata is required for the practitioners, as they might not be experts in the domain where they develop DL models. According to the practitioners that participated in the multiple case study, lack of metadata creates confusion and a poor understanding of the data. Further, the semantics of the data is often obscured due to poor organization, leading to ambiguities.

Implications: As most of the companies use supervised learning for training their DL models, lack of labeled data is considered as one of the biggest challenges. The impact of the absence of metadata challenge varies for different types of data. For instance, the dataset for building stock market price prediction will have opening price, closing price, quoted price, session price, etc. When the metadata is missing, it is hard for practitioners who develop DL

models to identify and distinguish different prices. On the other hand, if the dataset consists of images, audio, or video, metadata extraction methods can be employed to extract the labels for that dataset. Another associated challenge is that without metadata, it is difficult to analyze if some pattern makes sense or not. For example, a particular column representing the temperature of a location cannot be always zero, while the column indicating the value of the on/off switch can have zero all the time.

Empirical basis: Recommender system, wind power prediction system, house price prediction, melanoma detection, financial fraud detection, manufacturing system has encountered both lack of labeled data problem and absence of metadata problem. For instance, practitioners sought help from a doctor to label the skin lesions while developing a melanoma detection system.

4.1.2. Data granularity

Description: Data collected over a period is aggregated and stored in a database or data warehouse. Data aggregation may remove important data points that cannot be collected again. Thus, fine granularity or details in the data is lost through data aggregation techniques. Like in mobile networks, counter data is collected and aggregated to a value over 15 min, and it is stored. Because saving every second's data point is expensive.

Implications: As a result of data aggregation, a significant amount of information is lost, which could mean that even though a lot of data are in place while looking at the details, the granularity needed for a use case will not be there. Therefore, even if data is collected over ten years, the problem is still limited by data collection choices which are difficult to get around.

Empirical Basis: In our study, the recommender system case experience this data granularity problem. When the reviews from users are all logged for a long period and handed over for building recommender systems, but failed to log the user's identity, the data granularity is lost. Further, combining the data with another dataset on the user level becomes impossible. Wind power prediction systems also suffer from this challenge.

4.1.3. Shortage of diverse data samples

Description: Upon training, the deep neural network should be given all possible instances and varieties of data so that it will not fail on inputting unseen data in production. However, during data collection, many companies collect numerous abnormal samples and fail to collect the counterexamples of data and vice versa, leading to a class imbalance problem. The DL model needs to be trained with counterexamples as well. For instance, the company which did data collection for financial fraud detection only collected fraudulent samples and failed to save the non-fraudulent transactions in the dataset.

Implications: Financial fraud detection cannot be developed only with the samples of fraudulent transactions, the model needs non-fraudulent samples to distinguish between normal and abnormal financial transactions. DL models cannot learn the normal cases by themselves when only the abnormal samples are fed during the training. The models that are trained on such input data will produce weird outputs once deployed in production.

Empirical Basis: In our study, wind power prediction system, house price prediction system, melanoma detection, and financial fraud detection system suffers from this challenge. Melanoma detection is the use case that has the highest impact due to this challenge. Collecting malignant samples from patients is a difficult process for the data collection team.

4.1.4. Data sharing and tracking methods

Description: Sharing the collected data with the practitioners is required for implementing DL models. There is no defined channel or medium for sharing the collected data. According to the size of the data, companies choose different means of sharing. Some companies opt to share the data in the form of Excel files over email, FTP server, or even in the form of physical tapes. According to the practitioners, data for wind power prediction were given in the form of huge magnetic tapes, while the data for house price prediction was sent through email.

Implications: Two of the experienced practitioners identify data tracking as an important measure by which data quality can be assured. However, due to the tight limit on time and resources, often data tracking is not focused much or is kept at the least priority, leading to poor data quality.

Empirical Basis: All use cases except melanoma detection systems do not have data sharing and data tracking methods. Data pipelines are used for melanoma detection use cases, which enables sharing and tracking of data.

4.1.5. Data storage complying to GDPR

Description: DL systems are powerful and have the potential to memorize every piece of information given to it. Thus, the amount of training data has the biggest impact on the performance of the model. General Data Protection Regulation (GDPR) is a regulation in EU law to protect online personal data. GDPR is a set of legislative rules which impose restrictions on the processing and storage of information. Major companies that focused on collecting and maintaining datasets can build better DL models to a certain extent. The problem with small-scale companies is that they do not have clear knowledge on how to collect and store data complying with the rules of GDPR, and there is no framework or protocol to help them to do data storage efficiently.

Implications: In such cases, a certain percentage of revenue needs to be paid as a penalty for not following the regulations of GDPR, which end up in the deletion of a huge portion of data they collected over time.

Empirical Basis: Recommender system's use case encountered this challenge and lost a considerable amount of data as they had to delete the data to comply with the GDPR. Even though this is a problem experienced by only one case in the entire study, it is important as it has significant legal and financial complications involved.

To summarize, data management challenges at the data collection step are lack of data labeling techniques, unavailability of fine-grained data, shortage of diverse samples, lack of data sharing and tracking methods, and data storage guidelines following GDPR.

4.2. Data exploration

Data exploration, or exploratory data analysis (EDA), is a process to analyze and understand the data with statistical and visualization methods. This step helps to identify patterns and problems in the dataset, as well as for deciding which model or algorithm to use in subsequent steps. Statistical understanding, data deduplication complexity, and data heterogeneity are the major challenges encountered at this phase of the data lifecycle.

4.2.1. Statistical understanding

Description: When confronted with data that needs to be analyzed, the first step is to carefully identify the distribution of data. Statistical understanding is much required for determining the distribution of data. Even with sufficient knowledge in statistics, it is challenging to identify the distribution of data. The normal

distribution or Gaussian distribution is that nice, familiar bell-shaped curve. But, data comes from a range of devices out in the wild, and there is no point in assuming an easy to handle normal distribution.

Implications: For instance, consider an image processing application, to model the pixel values efficiently, the assumption of Gaussian distribution is inaccurate as it violates the boundary properties. In such cases, models like BMM (Beta Mixture Model) should be used. Without clear knowledge of statistical distributions, it will become difficult to model the distribution.

Empirical Basis: Practitioners mentioned that they have encountered this challenge while developing Wind power prediction systems, financial fraud detection systems, and manufacturing systems.

4.2.2. Data deduplication complexity

Description: A dataset often has a lot of duplicates, some with slight variations and some exact copies. Consequently, analyzing the dataset for duplicates and de-duplication is a complex task. For example, consider a song recommender system trained on a dataset of songs. If you take a random song, there can be 200 versions of the same song with slight variations in it, but it is more or less the same song. If the model is trained with such a dataset, the result may turn out horrible, such that it may recommend 50 copies that sound more or less the same. In such cases, de-duplication becomes complex. Because if the dataset has 100,000,000 songs, you need to compare a song with every other song in the dataset. Therefore, it is a quadratic complexity of that problem.

Implications: It is impossible to do from a time point of view in a single machine, and it is required to run it on hundreds and hundreds of machines.

Empirical Basis: According to the practitioners, they have encountered this challenge for all use cases presented in this study. Especially, with the melanoma detection model, it was almost impossible for them to deduplicate the images in the dataset.

4.2.3. Data heterogeneity

Description: Different data sources may offer conflicting information. Moreover, rapidly growing multimedia data from the Web and mobile devices consists of a huge collection of still images, video and audio streams, graphics and animations, and unstructured text, each with different characteristics. The major challenge here is to find a method that can resolve the conflicts and fuse the data from different sources effectively and efficiently. The majority of current DL algorithms are evaluated on bi-modalities (i.e., data from two sources). However, format, size, and encoding techniques vary from data to data. A single dataset itself may have data in audio, video, and text formats. If a dataset containing only textual data is examined, some text will be in UTF-8, others in UTF-16, some in CSV, comma-separated format, others in tab-separated format, some with HTML code embedded in the actual text, and others with something strange like placeholders embedded inside the actual national language text.

Implications: It is required to invest a significant amount of time and effort in just transforming the text into a uniform format and coding for the data.

Empirical Basis: All six use cases studied here have encountered this challenge. Moreover, the system performance decreases for significantly enlarged modalities. Furthermore, practitioners lack the idea about levels in DL architectures appropriate for feature fusion with heterogeneous data.

Statistical understanding, data de-duplication, data heterogeneity are the main data management challenges encountered in the data exploration phase.

4.3. Data preprocessing

Data preprocessing is an integral step in DL, as the quality of data and the useful information that can be derived from it has direct impact on the model's learning ability. Therefore, data must be preprocessed before feeding it into the model. Dirty data, managing sequences in data, and managing categorical data.

4.3.1. Dirty data

Description: Raw data is typical with imperfections like missing values, wrong values, and ill-formatted values. These unclean or noisy data are known as dirty data. Deep neural networks are good at deriving patterns from the given input. So, it is dangerous to feed noisy data to the DL models. Also, the DL experts might not be experts in the domain, and so they are unaware of what needs to be filled when there are missing values and how to identify the wrong or ill-formatted values. For example, if there is a column for age and some values are missing. The system is supposed to make predictions based on each user, and you do not have the age for 10% of them. That column can be filled out with the average or minus one. To fill the column, it is required to know what the column is meant to be and what can be filled in to replace the missing/wrong/misformatted values. All practitioners mentioned the unclean data issue in all the cases they handled, and in most cases, discussion with the people who collected the data was the only practical solution.

Implications: Dirty data impacts are related to error type and error rate. Thus, the error rate of each error type in the provided data should be necessarily detected. Testing accuracy decreases with increased noise in data.

Empirical Basis: According to the input from practitioners, all datasets have the problem with dirty data, and it takes an enormous amount of time to fix the noise in the data.

4.3.2. Managing sequences in data

Description: Metadata management should be considered with equal importance in managing the sequences in data. Storing the sequencing data alongside the contextual metadata is a bit challenging, especially when the data quantity is too large. For instance, for chronological data, there is a time series that needs to be divided chronologically somehow, so we do not end up predicting the past. Missing the time steps in the sequence is another associated challenge. i. e., data has variable-length sequences by definition. Those sequences with fewer time steps may be considered to have missing values.

Implications: Short gaps in the sequential data can be imputed using the before and after data present in the sequence. However, when the gap increases, it consists of more contiguous missing values, and consequently, the amount of information in the sequence is often not sufficient to impute the gaps.

Empirical Basis: House price prediction system and wind power prediction system have the problem with missing time steps.

4.3.3. Managing categorical data

Description: Variables containing label values rather than numerical values are referred to as categorical data. Nominal, ordinal, interval and ratio are examples of categorical variables. County – Tuscaloosa, Mobile, Walker, and so on are examples of nominal variables with attribute values that have no natural order. The difference between values (e.g., Letter Grade – A, B, C, D, F) does not have a natural order in ordinal variables. The difference between the two values is meaningful (e.g., Age of Driver – 22–24, 25–34, 35–44, etc.). Interval variables are constructed from intervals on a continuous scale. A ratio variable has all the properties of an interval variable, but it also has a distinct definition of 0.0. (e.g., Weight). DL models cannot operate

on label values as it requires all input variables in the numeric form. Even though one-hot encoding is used very frequently, it can be frustrating during implementation.

Implications: When there are thousands of categories, the complexity increases. For example, if there is text data that needs to be cleaned up and converted to numeric form, then it might not be possible to do it with a laptop or even a big server. There are core systems like Hadoop, Spark, or Google Data Flow where big data processing can be done. However, it is still very dependent on the person doing it, what they are comfortable with, and also the data, how big is it, how difficult is it, and what needs to be done with it. There are no predefined sets or standards to handle this.

Empirical Basis: House price prediction system, financial fraud detection system have more categorical data compared to the other use cases.

To summarize, dirty data is the major challenge in this phase, and reducing its effect needs an enormous amount of time and effort from the practitioners. Managing categorical data, as well as sequential data, are the other two challenges encountered in this phase.

4.4. Dataset preparation

After the data is preprocessed, the dataset is split into two parts: training and testing datasets. The training dataset is again split into training and validation datasets. The ratio of these two splits varies according to the size of the dataset, developer, and type of dataset. Nevertheless, the typical practice is to split the dataset in an 80:20 ratio for training and testing respectively.

4.4.1. Data dependency

Description: Data leakage is the challenge of not splitting the training and validation/test dataset properly so that the training data for the model happens to have the data which needs to be predicted. For instance, data leakage happens when the same data instance occurs in both training and testing datasets.

Implications: Data leakage hides the actual performance of the model and when it is exposed to new and unseen data, the performance will not be as expected. So proper attention should be taken while splitting the dataset. Based on the study, we could infer that checking the data distribution is not always a solution to reduce data dependency.

Empirical Basis: All use cases discussed in this paper have encountered this challenge.

4.4.2. Data quality

Description: Quality of data is crucial, as poor quality data can cause severe performance degradation and exaggerated results. Data consistency is one of the factors deciding the quality of data. However, consistency is hard to achieve the target in many applications. For example, based on our study, the images collected from the hospitals are all taken in different conditions with different lighting. Accuracy, completeness, validity, and timeliness are some other factors that ensure the quality of data. However, there is no exhaustive list of factors that should be checked to ensure the quality of data, which is challenging. Although the dirty data problems are fixed in the data exploration phase, DL experts mentioned that they recheck the data quality during the dataset preparation stage. Data exploration primarily looks for dirty data problems with the help of domain experts. However, in the dataset preparation phase, it is usually the DL expert looks if he/she has sufficient quality data before feeding it as input to the model.

Implications: DL models learn by adjusting their internal parameters with massive quantities of training data until they can

consistently discern comparable patterns in data they have never seen before. Therefore, a deep learning model is acutely sensitive to the quality of the data. Because of the huge volume of data required, even relatively small errors in the training data can lead to large-scale errors in the system's output.

Empirical Basis: Data quality is a typical challenge faced by all practitioners, and all the datasets discussed here have a problem with data quality.

To summarize, data quality is the main challenge experienced in this phase. Identifying what all factors determine data is a challenging task, and it, in turn, creates trouble while establishing tests for checking quality.

4.5. Data testing

Data testing is the phase in which practitioners decide whether to retrain the model or not. When the test cases fail, it indicates that there is a possibility of change in the underlying data. According to the experts participated in the interview, data testing is done at least once before the model deployment. After the deployment, during the continuous monitoring if data drift is encountered, data testing is done again to check if retraining is required or not. Therefore, data testing is a step that is conducted multiple times in the data life cycle.

4.5.1. Expensive testing

Description: Testing the data is a critical step that ensures the data quality and reduces the possible occurrence of defected data that affects the efficiency of the process. Absent, obsolete, or wrong test data may prevent the practitioners from executing the test cases or produce unreliable test results.

Implications: Data testing is highly expensive in the sense that it requires a lot of effort and time to define and automate test-cases specific to DL models. It is pretty hard to do regression testing on data as the data is collected from users out in the wild, where exerting control is impossible.

Empirical Basis: Practitioners who work with recommender systems, melanoma detection system and manufacturing systems has mentioned this challenge.

4.5.2. Data management tools

Description: Tooling is a challenge in most of the phases of the data lifecycle. Although tooling is a challenge experienced at most of the data lifecycle stages, many of the tools are under development. For instance, there exists data cleansing tools such as TIBCO Clarity, Data Wrangler etc. helps with cleaning the dirty data. However, tools for data testing is still an unexplored area. The major advantage of conventional software systems is that there exists a large variety of tools, especially for testing. As DL is a recently emerged approach, tools for testing such models are yet to be developed.

Implications: With the help of data management tools, practitioners can easily develop and deploy DL models.

Empirical Basis: Although cloud has good quality tools and services, none of the companies who have participated in the study can use that due to the policy restrictions. Therefore, all the cases included in our study experience tooling problems.

The main challenge in this phase is the lack of tools for creating test cases for DL systems. Existing methods are highly expensive.

4.6. Model deployment

The process of running an application on a server or device is known as model deployment. All the stages, processes, and activities required to make a DL available to its intended users are included in the model deployment. Data extraction methods and overfitting are the two challenges encountered in this phase.

4.6.1. Data extraction methods

Description: Training-serving skew is a typical problem encountered when running DL models in production, where the data encountered at serving time is significantly different from the data used to train the model, leading to reduced prediction quality. **Implications:** For example, Google once built a system called quick access in Google Drive which recommends a list of documents to open. When the system was built, they first extracted data and made a training dataset, trained a model on it, and did the evaluation which looked great. So, they put it in production, and it did not work. When they investigated, they discovered that they had a specific pipeline for extracting data for training, but when they deployed it in production, they had data extracted from an API, which did not match the extraction they had for training. Thus, some additional transformation happening in the API prevented the model from working as expected.

Empirical Basis: This challenge was encountered during the deployment of all the DL models discussed in this paper. Practitioners mentioned that the main reason for the challenge is the lack of tracking of data extraction methods. When an expert is replaced by another one, the new expert might not exactly know what data extraction method was used during the development phase and use a different extraction method during deployment, leading to training serving skew.

4.6.2. Overfitting

Description: Overfitting is the situation when a deep neural network memorizes and fits itself so closely with the training set that it loses the capability to generalize and make predictions for new and unseen data.

Implications: An overfit model can cause the regression coefficients, p-values, and R-squared to be misleading. Because an overfitted model will be tailored to the specific data points that are included in the training sample and not generalizable outside the training sample. Therefore, it gives poor performance to unseen data.

Empirical Basis: For instance, in melanoma detection use case where tabular data is used along with images, it turned out that the model was just learning the ID number of a certain hospital, and that hospital was a popular hospital to which the more severe cases were sent. Thus, the model was not learning anything from the images, rather it was just learning that the patients in that hospital are more likely to be sick, which is because they were sent there.

Data Extraction methods and Overfitting are the two main problems encountered in this phase of the data pipeline. Usage of different data extraction methods while training and testing create problems. Overfitting is a common challenge with machine learning as well as DL systems. Overfitting challenge has solutions such as cross validation and regularization when detected during model training. However, during model deployment and post deployment, it is hard to solve.

4.7. Post-deployment

Monitoring the models that are deployed in production is an important task in the data lifecycle. Because, change in the data source, distribution, data drifts, etc. can cause serious performance degradation. Therefore, these are considered as challenges after the model deployment.

4.7.1. Changes in data sources and distribution

Description: When a certain problem is modeled, a distribution is postulated based on the data available at that time. However, consistency in data distribution cannot be expected all the time. Consider the house price prediction system in our study, which

is trained on historical real-estate data. When some sudden environmental disaster or society-wide effect takes place, the usual distribution will be disturbed, and the trend in data changes. Deep neural networks may not always be able to handle new data distributions when they appear. An abrupt change in the data source can sometimes result in unexpected and undesirable effects.

Implications: Change in data distribution has a far-reaching impact on any DL model. For example, when building the DL model, data that arrived before a change can bias the models towards characteristics that no longer hold.

Empirical Basis: For instance, the distribution of the price of houses shows abnormal deviation after a natural calamity. It is the same with data for wind power prediction systems as well. As a result, an already deployed well-performing model in production may not perform well after the change in data distribution.

4.7.2. Data drifts

Description: Data drifts are also known as data shifts that happen over time. When data shifts happen, DL models may deliver weird and erroneous results. Consider systems, such as mobile interactions, sensor logs, and web clickstreams. Whenever the business tweaks or updates happen, the data those types of systems generate changes continuously. The sum of these changes is data drift. Other common examples of structural drift are fields being added, deleted, and re-ordered, or the type of field being changed.

Implications: When drift in a model is detected, the next step is identifying which features in the data are causing the drift. It is possible that some features have drifted but have not created a significant change in the model because these features are not very important. Identifying the feature that causes the drift and is of great importance to the model, is crucial to the performance of the model and should therefore receive better attention when retraining the model.

Empirical Basis: For example, data for case E, which is a financial fraud detection system, have this challenge. To support a growing customer base, a bank adds leading characters to its text-based account numbers. This kind of data change causes the bank's customer service system to combine data related to bank account 00-56789 with account 01-56789. All practitioners agree that most of the cases that they handle are subjected to this challenge.

4.7.3. Feedback loops

Description: Feedback loops are sometimes beneficial and at times detrimental. For instance, feedback loops are inherent to the recommendation systems. Because, the data collected will be mostly from the customers and if good suggestions are given on what to buy, of course, the customers will buy more. As a result, the model sort of reinforces itself.

Implications: These feedback loops give rise to the “echo chambers” or “filter bubbles” that have user and societal implications. Systems that lead to a self-reinforcing pattern of narrowing exposure and shift in user’s interest are often denoted as “echo chamber” and “filter bubble”.

Empirical Basis: Feedback loops are not a typical challenge with regression models or classification models. Therefore, only case A has encountered this challenge.

To summarize, the data management challenges that can be found at the post-deployment stage include the change in data sources and distribution, feedback loops, and data drifts. A degenerative feedback loop is a problem specific to recommender systems.

Table 4
Mapping between Data Lifecycle phase, Challenges and Potential Solutions.

Data lifecycle Phase	Challenge	Potential Solutions validated through case study
Data Collection	1. Lack of labeled data	1. Crowdsourcing 2. Active Learning 3. N-shot Learning 4. Unsupervised Learning 5. Semi-supervised learning 6. Self-supervised Learning 7. Help from Domain Experts
	2. Data Granularity	1. Lossless data aggregation 2. Synthetic data
	3. Shortage of diverse samples	1. N-shot Learning 2. Resampling 3. Synthetic data generation 4. Class weighting with data augmentation
	4. Data sharing and tracking methods	1. Data Pipelines
Data Exploration	5. Data Storage complying to GDPR	
	6. Statistical Understanding	
	7. Deduplication Complexity	
Data Preprocessing	8. Heterogeneity in data	1. Data Pipelines
	9. Dirty data	1. Data Cleaning 2. Data Scrubbing 3. Data Imputation
	10. Managing sequences in data	
Dataset Preparation	11. Managing categorical data	1. Learned Embeddings 2. One-hot encoding
	12. Data Dependency	
	13. Data Quality	1. Data Validation Frameworks 2. Data Linter
Data Testing	14. Tooling	1. Cloud based services
	15. Expensive Testing	
Deployment	16. Data Extraction Methods	
	17. Overfitting	1. Regularization 2. Cross Validation 3. Model simplification 4. Data Augmentation 5. Dropouts 6. Transfer Learning
	18. Data sources and Distribution	
Post-deployment	19. Feedback Loops	
	20. Data Drifts	

5. Solutions for data management challenges

This section summarizes the solutions to the challenges identified and described in the previous section. The solutions presented were derived from a systematic literature review and validated in the second round of multi-case study research. Further, Table 4 shows the mapping between the data lifecycle phase, challenges, and the identified solutions. Data management existed even before the emergence of DL and machine learning. Therefore, many of the data management challenges were solved in the context of big data analytics, data mining, machine learning, etc. On the other hand, some challenges are very specific to DL. For instance, categorical data challenge, tooling, etc. are very specific to DL. Although there exist solutions for a subgroup of data management problems, industrial practitioners still list them as challenges. Therefore, we did a second round of multiple case study analysis to explore the validity of identified solutions.

5.1. Data collection

Challenges such as lack of labeled data, data granularity, shortage of diverse samples, need for sharing and tracking techniques are the challenges encountered in this data lifecycle phase. Crowdsourcing, active labeling, semi-supervised learning, and self-supervised learning are some solutions that can address the lack of labeled data challenges. Few-shot learning, zero-shot learning, less than one-shot learning can be used to address both lack of labeled data challenge and shortage of diverse sample problems. Resampling with data augmentation, synthetic data and data augmentation with class weighting are some other solutions that help with the shortage of diverse sample problems. Data granularity can be addressed with lossless data aggregation techniques and synthetic data. Data pipelines can partially solve the challenge of data sharing and tracking techniques.

5.1.1. Crowdsourcing

Description: Crowdsourcing is a brilliant platform to outsource high-volume data labeling jobs to a flexible workforce (Malone et al., 2009). Crowdsourcing usually achieves its purpose by combining the responses of numerous annotators on the same sample. For instance, Nowak et al. (Nowak and Rüger, 2010) considered the problem of multi-label image annotation by experts and ordinary annotators from the MTurk crowdsourcing platform. After calculating various agreement statistics, the authors determined that if annotators are specialists, several labels for an object are redundant with high-quality recommendations for annotators. On the contrary instance, despite annotators' excellent accuracy, a dataset produced by multiple annotators doing a single task is of much greater quality.

Crowdsourcing is extensively utilized to solve problems that are not accessible to automatic calculations and necessitate human intervention. Three things hindrances affect the efficiency of crowdsourcing platforms. The first is quality, which refers to algorithms that accurately discriminate between true and false labels. The second factor is the labeling costs: it is not always feasible to fix an issue by increasing the number of annotators per sample. The third factor is that, in some cases, labeling speed is critical; in these cases, task execution latency must be kept to a minimum (Gilyazev and Turdakov, 2018).

Challenges addressed: Lack of labeled data

Limitations: According to the practitioners, crowdsourcing has the following limitations.

- Poor quality data labels
- Lack of agility with the workforce or tools
- Need for reviewing and evaluating the quality of labels
- Crowdsourcing cannot work in the absence of metadata

Since manually validating the quality of submitted results is laborious, unethical personnel frequently take advantage of this limitation and submit low-quality responses. As a result, practitioners have stated that they require systems that can reliably evaluate worker quality, allowing for the rejection and filtering of low-performing workers and spammers. Further, when the annotators in a pool work more, gradually their efficiency decrease, affecting the quality of annotations. Consequently, more number of workers in a pool demand management, which is an overhead.

5.1.2. Active learning

Description: One method to make the data labeling process easier is to use active learning. The algorithm takes one example from the unlabeled set for each iteration, then passes it to the oracle (expert) for labeling, and the classifier is re-trained using the new set of training data. The labeled cases are chosen by an active learning algorithm and added to the training set. A learner typically starts with a small collection of labeled cases, selects a few informative instances from a pool of unlabeled data, and queries an oracle for labels (e.g., a human annotator). The goal is to reduce the total labeling cost to train a reliable model. (Settles et al., 2008). Active learning approaches seek out the most informative examples for the classifier, resulting in a substantial reduction in the amount of labeled data, as proven both theoretically and empirically. (Settles, 2009). Further, it helps the practitioners to identify the part of the data that needs to be labeled to achieve maximum benefit. For instance, more examples near the classification boundary help more compared to the labeled instances that lay far away from the decision boundary. Active learning assumes that labeling is performed by just one expert (oracle) at any given instance. However, it is often required to parallelize the load, thus allowing the expert to label more samples. To

connect businesses and employees, crowdsourcing services such as Amazon Mechanical Turk (MTurk), CrowdFlower, and Toloka are used. Any platform user can create a job, such as labeling a set of numerous samples. Another user (annotator), having found an interesting task, performs it for a certain fee, which is much lower than the cost of involving an expert.

Challenges addressed: Lack of labeled data

Limitations: Based on the inputs from the practitioners, active learning has the following limitations:-

- Low-quality labeling due to data ambiguity, poor guidelines for annotators, and lack of motivation or knowledge
- A fixed cost cannot be assumed for acquiring each label. For example, if labels are acquired by executing a biological experiment, then the cost of a query might be the price of the materials used.

5.1.3. N-shot learning

Description: A shot is simply a single sample that can be used for training. As a result, N-shot learning has N training instances. The "few" in the term "few-shot learning" normally ranges from zero to five, so zero-shot learning refers to training a model with no instances, one-shot learning refers to training a model with one example, and so on. Few-shot learning is defined as learning new concepts from only a few labeled examples (Triantafillou et al., 2017). K-shot N-way classification is the job of categorizing a data point into one of N classes when only K instances of each class are available to guide the decision. This is a challenging situation that needs methods that are different from those used when there is a lot of labeled data for each new concept. Deep learning algorithms rely on large datasets and are prone to overfitting due to a lack of data. However, expecting many examples for learning a new class or concept is unrealistic and undesirable, making few-shot learning a critical issue to handle. Few-shot learning aims to get as much information out of each training batch as possible, which is especially critical when the amount of data available for learning each class is restricted (Snell et al., 2017). The ability to classify instances of an unseen visual class, called zero-shot learning (Socher et al., 2013). Zero-shot learning can be used for products or activities that lack labeled data and new visual categories, such as the latest electronics or automobile models. The goal of zero-shot learning is to recognize items that were not seen during training (Xian et al., 2017). Some common zero-shot splits may regard feature learning as a separate stage from training, necessitating the creation of additional dataset splits. Furthermore, in a zero-shot learning context, separate training, and validation class split is a crucial component of parameter adjustment. Image classification systems in real-world applications do not have advanced knowledge of whether a new image corresponds to a seen or unseen class. As a result, generalized zero-shot learning is appealing from a practical standpoint (Romera-Paredes and Torr, 2015). The model must learn a new class from a single example in one-shot learning, which is an extreme variant of few-shot learning. Models learn N new classes given just M less than N examples in a 'less than one-shot learning assignment, which can be accomplished with the use of soft labels (Sucholutsky and Schonlau, 2020). The essential premise of less-than-one-shot learning is that after a few categories have been learned the hard way, some information from that process can be abstracted to make learning subsequent categories more efficient. In other words, rather than starting from scratch to learn a new or unfamiliar category, make use of existing information (Fei-Fei et al., 2006). However, n-shot learning is also not free from limitations.

Challenges addressed: Lack of labeled data

Limitations: Based on practitioners experience, n-shot learning has the following limitations:-

- Lack of generalization
- Hinders the classification ability of the model when the images have noise
- Not a popular technique in industry

5.1.4. Unsupervised learning and semi-supervised learning

Description: To address the issue of annotation, we need a method that reduces the requirement for annotation, as target annotation is typically costly. Unsupervised learning (UL) is a machine learning algorithm that works with unlabeled datasets. Cluster analysis is most typically used to identify hidden patterns in huge unlabeled datasets. UL can deduce the structure of data that has not been labeled (Celebi and Aydin, 2016). However, they necessitate some human interaction when it comes to validating output variables. An unsupervised learning model, for example, can detect that online buyers frequently purchase groups of products at the same time. A data analyst would need to confirm that grouping makes sense for a recommendation engine. Feature learning settings that are unsupervised, depends on the unlabeled data. The self-taught learning to set is a more general and powerful option, as it does not require your unlabeled data to come from the same distribution as your labeled data (Triguero et al., 2015). The semi-supervised learning (SSL) paradigm has received a lot of attention in a variety of domains, from biology to Web mining, where getting unlabeled data is easier than getting labeled data since it takes less effort, expertise, and time (Zhu and Goldberg, 2009). Self-labeling approaches are used in semi-supervised algorithms. Self-labeled techniques are a viable and promising category of strategies for utilizing both types of data, and they are strongly tied to other methods and difficulties. Single-classifier methods and multi-classifier methods are two types of self-labeled approaches. In semi-supervised learning, numerous labeled instances are typically required to train a weakly effective predictor, which is then utilized to exploit the unlabeled data (Zhou et al., 2007). However, there may be very few labeled training examples in many real-world applications, which makes the weakly useful predictor difficult to generate, and therefore these semi-supervised learning methods cannot be applied. Further, they have limitations in extracting the most confident predictions from the learner (Zhai et al., 2019).

Challenges addressed: Lack of labeled data

Limitations: According to the practitioners, these learning methods have limitations, such as:

- The spectral classes do not necessarily represent the features in UL.
- UL does not consider spatial relationships in the data.
- UL can take time to interpret the spectral classes.
- Iteration results are not stable for semi-supervised learning.
- Semi-supervised learning does not apply to network-level data.
- Semi-supervised learning has low accuracy.

5.1.5. Self-supervised learning

Description: Self-supervised learning is a broad framework for learning that relies on surrogate (pretext) tasks that can be created using solely unsupervised data. A pretext assignment is created in such a way that completing it necessitates the acquisition of a useful visual representation (Zhai et al., 2019). Self-supervised learning attempts to overcome these restrictions by learning image representations directly from pixels, rather than depending on pre-defined semantic labels (Misra and Maaten, 2020). This is frequently accomplished using a pretext task that involves applying a transformation to the input image and requiring the learner to anticipate transformation properties from the produced image. Rotations, affine transformations, and jigsaw transformations are examples of transformations. Without

the use of labels, self-supervision produces effective representations for downstream tasks. These methods outperform systems that merely learn visual representations from unsupervised images. Self-supervised learning models, on the other hand, perform poorly compared to fully supervised learning models, and they are frequently not considered advantageous beyond obviating or minimizing the need for annotations (Hendrycks et al., 2019).

Challenges addressed: Lack of labeled data

Limitations: Practitioners mentioned that self-supervised learning had the following limitations:

- The reason behind its limited usage is mostly due to its novelty
- Building models can be more computationally intense
- Inaccurate labels may cause inaccurate results

5.1.6. Help from domain experts

Description: Domain experts to label data is one of the labeling techniques proposed in the literature (Shi et al., 2008). A domain expert possesses knowledge of the domain in addition to the ability to label instances. This knowledge can be used to identify areas of the feature space that are inaccessible, missing features that are required to divide classes, or features that are not required for the task at hand (Holmberg et al., 2020). Domain experts operate in the business domain. Their environment is made up of real-world business transactions and interactions, and a major portion of the knowledge they rely on has built up organically in the form of skill and experience, which they apply in a variety of ways (Viaene, 2013). Domain experts are especially useful in domains/use cases where label quality is critical. In reality, this frequently leads to a trade-off between cost and quality: one seeks to make the most of expensive domain specialists while also maximizing the use of low-cost crowd annotations (Nguyen et al., 2015).

Challenges addressed: Lack of labeled data

Limitations: Based on the case study, we understood that this solution has the following limitations:

- Not applicable for all sets of problems. For instance, hiring a doctor to annotate the inner body details from a surgery video might become a whole day process.
- Time-consuming and expensive

5.1.7. Lossless data aggregation

Description: To combat the challenge of the data granularity problem, one of the naive techniques is lossless data compression. Data compression is a common practice (Lin and Kolcz, 2012), especially when data from multiple sources are collected. BZIP2 based on Burrow Wheelers Transform (BWT) (Burrows and Wheeler, 1994) and GZIP based on LempelZiv (LZ) (Ziv and Lempel, 1977) are two popular and successful lossless text compression schemes widely used for the compression of data files. These techniques can be directly applied to time-series data. Plane fitting is a method used in Google for aggregating data obtained through laser range scans which accurately measure the depth, the two sides, and the front of the vehicle (Anguelov et al., 2010). Another common technique is to have a separate raw data storage where data files are stored before aggregation (Raj et al., 2020). However, the data storage space limitation makes this approach less acceptable for many small-scale companies.

Challenges addressed: Data granularity

Limitations: Practitioners that participated in the study said that lossless data aggregation has the following limitation:

- Hard to retain the original amount of data/details without loss during aggregation

5.1.8. Resampling and synthetic data generation

Description: Up-sampling and down-sampling are two methods that can be used to re-balance the class distributions, thus solving challenges with the imbalanced dataset. With synthetic data generation, artificial objects similar to the minority class are introduced into the dataset. SMOTE (Chawla et al., 2002), improved alternatives such as ADASYN (Chen et al., 2010) or RAMO (He et al., 2008) are some oversampling techniques. Down-sampling is the least preferred solution, as it may remove valuable samples in the dataset. However, running too many iterations of these leads to problems such as class distribution shift (Krawczyk et al., 2012). Besides the application of data level techniques, it is possible to choose algorithms such as decision trees that can perform well with the imbalanced dataset. Furthermore, performance evaluation metrics like precision, recall, and F1-measure can be used to reduce the impact of an imbalanced dataset problem. Upsampling with data augmentation is the most used industrial practice to mitigate the challenge of imbalanced datasets.

Challenges addressed: Shortage of diverse samples

Limitations: Based on the input from the practitioners, resampling and synthetic data generation has the following limitations.

- Oversampling increases the number of training examples, thus increasing the learning time.
- Oversampling makes exact copies of existing examples, likely leading to overfitting.
- Undersampling discards potentially useful data
- While synthetic data can imitate many properties of original data, it cannot exactly copy the original content.
- Synthetic data may not often cover the corner cases like authentic data
- Synthetic data needs a verification server

5.1.9. Class weighting with data augmentation

Description: Manipulation of data, such as weighing data examples or adding new instances, is becoming more popular as a solution to the problem of an imbalanced dataset. Data augmentation, for example, expands the data size by applying label-preserving changes to original data points; class weighting assigns significant weight to each instance to modify its effect on learning, and data synthesis generates entire synthetic cases. The use of data augmentation methods for time series data is fraught with challenges. To begin with, current data augmentation approaches do not completely use the intrinsic features of time series data. The so-called temporal dependency is a distinctive attribute of time series data (Wen et al., 2020). Time-series data, unlike image data, may be converted in the frequency and time-frequency domains, allowing effective data augmentation methods to be designed and implemented in the changed domain. When modeling multivariate time series, we must include the possibly complex dynamics of various variables across time, which makes things more complicated. As a result, just using image and audio processing data augmentation methods may not result in genuine synthetic data. Second, data augmentation techniques are task-specific. For example, data augmentation strategies that are appropriate for time series classification may not be appropriate for detecting time series anomalies (Hu et al., 2019). These two techniques of manipulation perform in diverse contexts: augmentation outperforms weighing when only a small quantity of data is available, but weighting outperforms augmentation when dealing with class imbalance issues. As a result, depending on the application parameters, the type of modification changes.

Limitations: Based on the input from the practitioners, class weighting with data augmentation is the most effective technique

to combat a shortage of diverse samples and class imbalance. They did not mention any limitations for this solution.

Challenges addressed: Shortage of diverse samples and class imbalance

5.1.10. Data pipelines

Description: Data pipelines are complex chains of interconnected activities that start with a data source and end in a data sink. In a data pipeline, the output of one component becomes the input to the other (Van Alstyne et al., 2016) and allows smooth, automated flow of data from source to destination. Data pipelines enable data sharing between two companies or organizations within a company. The ultimate destination of a data pipeline need not be data storage. Instead, it can be any application such as a visualization tool (Matheus et al., 2018; Stadler et al., 2016), Machine Learning(ML) models (Gautam and Yadav, 2014; Tanzil et al., 2017) or Deep Learning(DL) models (Covington et al., 2016; Deng et al., 2013). The data pipeline's components can automate the operations of extracting, processing, integrating, validating, and loading data (Sun et al., 2018). Data pipelines can process different types of data such as continuous, intermittent, and batch data (Goodhope et al., 2012). Moreover, data pipelines eliminate errors and accelerate the end-to-end data processes, which in turn reduces the latency in the development of data products. Monitoring the data pipeline activities can solve the problem of data tracking (Munappy et al., 2019).

Challenges addressed: Need for data sharing and tracking methods, data heterogeneity

Limitations: According to the practitioners, data pipelines has the following issues.

- Building data pipelines for a use case demands an unreasonable amount of time and effort
- Need to identify the activities which consume data, the output of each activity, order of execution, monitoring methods, intermediate storages, where to place the storage in the pipeline, data collection method, etc. which varies between use cases

5.2. Data preprocessing

Dirty data, managing sequential data and categorical data are the main challenges in the data preprocessing phase. Dirty data is an umbrella term used for a collection of problems and there exist solutions such as data scrubbing, data cleansing, and data imputation. To manage categorical data, one-hot encoding, label encoding, and embedding vectors are the major solutions. However, managing sequences in data are not explored in the literature and practitioners are struggling with this challenge, although it is not common in all the use cases.

5.2.1. One-hot encoding

Description: Scaling converts categorical data to numerical data by turning one numerical data type into another numerical data form during coding (Zhang et al., 2003). The two most common scaling methods for converting categorical data into numerical form are one-hot encoding and label encoding. When categorical data has no link between categories, a one-hot encoding is appropriate. Each category variable is represented by a binary vector with one element for each unique label, with the class label set to 1 and all other elements set to 0. One of the advantages of one-hot encoding is that the result is binary rather than ordinal, and everything is stored in an orthogonal vector space. The drawback is that with large cardinality, the feature space can quickly expand, resulting in the curse of dimensionality. One-hot encoding produces billions of trailers and billions of one-hot vectors, which is challenging to manage as categorical

data grows. Furthermore, the mapping is entirely uninformed: “similar” categories in embedding space are not placed closer to each other.

Challenges addressed: Managing categorical data

Limitations: According to the practitioners, this technique has the following limitations:

- Curse of dimensionality means that the error increases with the increase in the number of features
- Need for Principal Component Analysis
- Poor Scalability
- Natural order is lost and so is the relationship between each unique category

5.2.2. Learned embeddings

Description: A learned embedding (vectors of numbers), typically known as an “embedding”, is a distributed representation for categorical data. Each category is assigned to a unique vector, and the vector’s attributes are changed or learned as the neural network is trained. The vector space acts as a projection of the categories, allowing closely related categories to naturally cluster together. This has the advantages of an ordinal relationship in that any such relationship can be learned from data, as well as a one-hot encoding in that each category has a vector representation. The input vectors are not sparse, unlike one hot encoding (do not have lots of zeros). The disadvantage is that it necessitates learning as part of the model and the creation of many more input variables (columns) (Guo and Berkahn, 2016). Embeddings not only save memory and speed up neural networks as compared to one-hot encoding, but they also show the intrinsic features of categorical variables by mapping similar values near together in the embedding space. One of the solution’s disadvantages is that the original data is replaced by a similarity matrix in the first phase, and dimensionality is decreased using an embedding in the second. The computation of a distance matrix can take a long time, depending on the data and distance function. An efficiently constructed distance function may be able to mitigate this. Furthermore, some embeddings are hypersensitive to noise in data. This may be mitigated by additional data cleaning. In addition, some embeddings are sensitive to the choice of their hyperparameters. This may be mitigated by careful analysis or hyperparameter tuning.

Limitations: According to the input from the practitioners, learned embeddings is the most effective technique to combat the categorical data challenge.

Challenges addressed: Managing categorical data

5.2.3. Solutions to dirty data problem

Description: Dirty data problem is an umbrella term that includes various data problems such as incompleteness, uniqueness, incorrectness, inconsistency, inaccuracy, Kim et al. (2003) etc. Data scrubbing is the procedure to modify or removing incomplete, incorrect, inaccurately formatted, or repeated data in a dataset to improve consistency, accuracy, and reliability. Data cleaning is a process of tidying up the data, largely involving correcting or deleting obsolete, redundant, corrupt, poorly formatted, or inconsistent data. Data professionals do the actual cleaning, checking the dataset, and making corrections and edits as needed. Data scrubbing is a subset of data cleaning. Imputation is a powerful method used to solve the missing data problem. Imputation can be based on statistical methods as well as machine learning-based methods. K. Lakshminarayan et al. Jerez et al. (2010) has applied machine learning for implementing data imputation. Statistical methods include mean hot-deck and multiple imputation (Lakshminarayan et al., 1996). Data linter (Hynes et al., 2017) is a new class of tools that automatically inspects ML data sets to identify potential issues in the data.

Challenges addressed: Dirty data problems such as inconsistency, incompleteness, inaccuracy, redundancy, ill-formatted data.

Limitations: According to the practitioners, the solutions to the dirty data challenge is still not a well-explored area and have the following limitations.

- Hard to anticipate all potential data problems in a real-world context where data changes rapidly.
- Does not preserve the relationships among variables
- Leads to an underestimation of standard errors
- Time consuming and expensive

5.3. Data preparation

Data dependency and data quality are the two challenges encountered in the data preparation phase. To combat the challenge of data quality, data validation frameworks and data linter are used in practice. However, these are solutions are not applicable for image data.

5.3.1. Tools for testing data quality

Description: Data linter (Hynes et al., 2017) is an ML-based tool used in Google for detecting lints which can be classified into three high-level categories: outliers, packaging errors and miscodings of data. Data validation at the major steps of the data processing pipeline can be a possible option to detect and catch the errors affecting the data quality. However, complete prevention of data quality problems is not possible as the data generation happens in distributed data sources where exerting control is not possible. Therefore, the only possible option is to detect the quality issues in the early stages and implement mitigation strategies to overcome the issues (Lwakatere et al., 2021). To detect the data quality issues, test cases should be written and executed, which is an expensive task.

Challenges addressed: Data quality

Limitations: According to the practitioners, the data quality testing tools have the following limitations.

- Incompatible with image data
- Impossible to predict all potential problems with data
- Expensive and time-consuming

5.4. Deployment

The deployment phase faces challenges like data extraction methods and overfitting. Overfitting can be solved using regularization, data augmentation cross-validation, model simplification, transfer learning, and dropouts.

5.4.1. Regularization and cross-validation

Description: Regularization is a strategy for improving model generalization by making minor changes to the learning procedure. As a result, the model’s performance on previously unseen data improves as well. Google shows that L1 regularization can increase performance for a few kernels, but degrade performance in larger-scale instances. L2 regularization, on the other hand, never affects performance and improves it significantly with many kernels (Cortes et al., 2012). Zhang et al. (2018) and Cliche (2017) describes the application of regularizer in Facebook and Bloomberg, respectively. DeCov is a specific strategy used in Microsoft and Facebook to avoid the overfitting of their ML/DL models (Cogswell et al., 2015). Cross-Validation is a powerful preventative measure against overfitting (Whittaker et al., 2010). Google, Facebook, Uber, and many more large-scale enterprises employ K-fold cross-validation as their most widely used techniques. Cross-validation permits hyperparameter adjustment

with the sole original training set, while keeping the test set as a completely unknown dataset for final model selection. Regularization, on the other hand, aims to lower the estimator's variance by simplifying it, which increases the bias and reduces the anticipated error. When cross-validation is used, the processing power required is also considerable. As a result, in the case of huge data sets, the time it takes to acquire feedback on the model's performance increases.

Challenges addressed: Overfitting of DL models

Limitations: According to the practitioners, regularization and cross-validation has the following limitations:

- Curse of dimensionality, which means that the error increases proportional to the increase in number of features.
- Cross Validation is computationally costly, requiring a lot of processing power.
- Cross-Validation drastically increases the training time

5.4.2. Model simplification

Description: When dealing with overfitting, the first step is to reduce the model's complexity. We can lower the network's complexity by simply removing layers or reducing the number of neurons (Fahland and Van Der Aalst, 2013). While doing this, it is critical to calculate the input and output dimensions of the various layers involved in the neural network. Without a general rule on the size of the neural network, re-architecture is difficult.

Challenges addressed: Model simplification is used to address the overfitting challenge

Limitations: According to the practitioners, this solution has the following limitations:

- No guidelines on how to perform model simplification
- Expertise on Neural Network architecture is required

5.4.3. Dropout

Description: Regularization approaches like L1 and L2 change the cost function to reduce overfitting. Dropout, on the other hand, affects the network as a whole. It removes neurons from the neural network at random throughout each iteration of training (Srivastava et al., 2014). When different sets of neurons are dropped, it is equivalent to training different neural networks. The different networks will overfit in different ways, so the overall effect of dropout will be to reduce overfitting. Based on the case study for validation, dropout can sometimes decrease the performance of the model.

Challenges addressed: Dropout is a regularization technique that prevents neural networks from overfitting.

Limitations: According to the practitioners, dropout has the following limitations

- Dropout can hurt the model performance when training time is limited or when the neural network is small relative to the dataset

6. Summarizing challenges and solutions

We conducted a multiple case study to validate the industrial applicability of the identified solutions. Based on the limitations of the existing solutions and availability, we have classified the challenges into four, as shown in Fig. 4. Challenges with working solutions is a category of challenges that needs improvements in solutions in terms of time and cost. The second category is the challenges with partial solutions, where the solutions can be used to solve a part of the problem. For example, the lack of labeled data can be solved by using crowdsourcing, active learning, N-short learning, etc. However, if the metadata itself is missing, it cannot be solved by using any of the listed techniques. Challenges



Fig. 4. Classification of challenges based on the availability of solutions.

with specific solutions is a third category where the existing solutions are not applicable for all use cases. i.e. the solutions are applicable only for specific DL use cases. For instance, a data linter and data validation framework can solve the problem of data quality. However, they are not applicable for the use cases that use image datasets like Melanoma detection. Finally, the fourth category of challenges contributes to the open research challenges, as the listed challenges have currently no available solutions.

6.1. Challenges with working solutions

Data granularity is a challenge for which we have identified solutions such as lossless data aggregation techniques and synthetic data. Practitioners mentioned that they need more efficient solutions in terms of performance, as the existing solutions often affect the performance of the model. Managing categorical data is also a challenge for which learned embeddings are an effective solution. Practitioners did not mention any limitations for the technique. Similarly, overfitting is another challenge in this category, for which there exist a variety of solutions and regularization is the most widely used technique by the practitioners. The shortage of diverse samples can be solved using class weighting with data augmentation.

6.2. Challenges with partial solutions

The need for data sharing and tracking methods and the lack of labeled data are the challenges that fall in this category. Lack of labeled data can be addressed using crowdsourcing, active labeling, etc. However, if the metadata is missing, it is often not possible to use any of the existing solution approaches. Similarly, data sharing and tracking can be partly solved by data pipelines. However, implementing data pipelines for each use case itself is identified as an expensive task.

6.3. Challenges with specific solutions

Data quality, dirty data, data heterogeneity, tooling are the challenges for which the solutions are not general. The tooling problem is to an extent solved for use cases that can avail cloud services. Similarly, data heterogeneity can be solved by defining data ingestion modules in the data pipelines. However, the practical difficulties of establishing secure data pipelines between two parties make this solution not applicable for a set of use cases like wind power prediction.

6.4. Challenges without solutions

Challenges in this category are more compared to the other three categories. None of these challenges has any available working solutions. Therefore, this category needs much attention.

The results of our study confirms the data collection and data quality challenges described by Whang et al. However, our study is a more extensive version with more challenges identified and categorized according to the data lifecycle stages. Moreover, our study identifies the solutions for the identified challenges and explains the reason for listing those challenges even with the existence of solutions in the literature. We also try to identify the least explored areas, such as data testing challenge, solutions to feedback loop challenge, statistical understanding etc. Another closely related study is software engineering challenges for deep learning, where the authors describe and categorize the challenges into development, production, and organizational challenges. Some challenges such as unintended feedback loops, monitoring and logging are also described in the study (Arpteg et al., 2018).

7. Conclusion and research implications

Data management challenges are an important part of the DL model development. However, the current body of literature neither discuss the challenges nor the solutions. The inability to formalize the solution for data management leads to a need for manual enforcement of them. The research presented here shows that this is an error-prone and time-consuming task that takes most of the effort of the data scientists and other practitioners working with data during the construction-intensive phases of a project. This problem exists in traditional Machine Learning-based development, as well, but it is more apparent and acute in DL. Because Deep neural networks has been able to learn the minute details from the input data. However, industries applying DL have not been able to completely utilize the potential of DL due to several reasons, and the inability to solve the data management challenges is predominant among them. The cases presented in this study shows that the data management, is a bottleneck in DL projects and demands a lot of human intervention which leads to operational errors, training-serving skew and performance degradation. This paper presents 20 data management challenges for DL and their categorization according to the data lifecycle phase. Further, we have identified

solutions for these challenges through a systematic literature review. We have also conducted a second round of interview with the same practitioners to understand the reasons for not using these solutions in practice. Based on the availability of solutions, we further categorized the challenges into four, namely challenges with working solutions, challenges with partial solutions, challenges with specific solutions and challenges without solutions. We believe that results of this study can be used by researchers to identify and solve the challenges that are unsolved, partially solved and specifically solved. Further, it can help the practitioners to explore the solutions that are not familiar, such as N-shot learning. The implications for researchers are that there is a need for further research to find solutions in such a way that they are both amenable to automatic enforcement on the detailed design and easy to understand and use by both data scientists and developers and work in practice. The implications for practitioners are that there are solutions in the literature which remains totally unexplored, and this needs to be taken into account during the development of DL systems. Although the data management is inherently a task for a relatively small group, it should be possible to delegate the existing solutions to a larger group. This would give time for the data scientists to concentrate on the core tasks: data exploring, learn new tools, and maintaining the DL systems.

CRediT authorship contribution statement

Aiswarya Raj Munappy: Conceptualization, Methodology, Validation/Verification, Investigation, Writing. **Jan Bosch:** Project administration, Validation/Verification, Investigation (partly). **Helena Holmström Olsson:** Project administration, Validation/Verification, Methodology. **Anders Arpteg:** Project administration, Validation/Verification. **Björn Brinne:** Project administration, Validation/Verification.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is in part supported by Vinnova, Sweden and by the Software Center. The authors would also like to express their gratitude for all the support provided by Peltarion.

References

- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., et al., 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint [arXiv:1603.04467](https://arxiv.org/abs/1603.04467).
- Anguelov, D., Dulong, C., Filip, D., Frueh, C., Lafon, S., Lyon, R., Ogale, A., Vincent, L., Weaver, J., 2010. Google street view: Capturing the world at street level. Computer 43 (6), 32–38.
- Arpteg, A., Brinne, B., Crnkovic-Friis, L., Bosch, J., 2018. Software engineering challenges of deep learning. In: 2018 44th Euromicro Conference on Software Engineering and Advanced Applications. SEAA, IEEE, pp. 50–59.
- Bahdanau, D., Cho, K., Bengio, Y., 2014. Neural machine translation by jointly learning to align and translate. arXiv preprint [arXiv:1409.0473](https://arxiv.org/abs/1409.0473).
- Baxter, P., Jack, S., et al., 2008. Qualitative case study methodology: Study design and implementation for novice researchers. Qual. Rep. 13 (4), 544–559.
- Beaufays, F., 2015. Google AI blog: The neural networks behind google voice transcription. <https://ai.googleblog.com/2015/08/the-neural-networks-behind-google-voice.html>. (Accessed 05 January 2020).
- Bengio, S., 2000. Modeling high-dimensional discrete data with. In: Advances in Neural Information Processing Systems 12: Proceedings of the 1999 Conference, Vol. 1. MIT Press, p. 400.

- Bickman, L., Rog, D.J., 2008. The SAGE Handbook of Applied Social Research Methods. Sage publications.
- Burrows, M., Wheeler, D.J., 1994. A block-sorting lossless data compression algorithm. *Digit. SRC Res. Rep.*
- Celebi, M.E., Aydin, K., 2016. Unsupervised Learning Algorithms. Springer.
- Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P., 2002. SMOTE: Synthetic minority over-sampling technique. *J. Artificial Intelligence Res.* 16, 321–357.
- Chen, S., He, H., Garcia, E.A., 2010. RAMOBoost: Ranked minority oversampling in boosting. *IEEE Trans. Neural Netw.* 21 (10), 1624–1642.
- Chen, X.-W., Lin, X., 2014. Big data deep learning: Challenges and perspectives. *IEEE Access* 2, 514–525.
- Cliche, M., 2017. Bb_twtr at semeval-2017 task 4: Twitter sentiment analysis with cnns and lstms. arXiv preprint arXiv:1704.06125.
- Cogswell, M., Ahmed, F., Girshick, R., Zitnick, L., Batra, D., 2015. Reducing overfitting in deep networks by decorrelating representations. arXiv preprint arXiv:1511.06068.
- Cortes, C., Mohri, M., Rostamizadeh, A., 2012. L2 regularization for learning kernels. arXiv preprint arXiv:1205.2653.
- Covington, P., Adams, J., Sargin, E., 2016. Deep neural networks for youtube recommendations. In: Proceedings of the 10th ACM Conference on Recommender Systems. pp. 191–198.
- Daily, M., Medasani, S., Behringer, R., Trivedi, M., 2017. Self-driving cars. *Computer* 50 (12), 18–23.
- Deng, L., Li, J., Huang, J.-T., Yao, K., Yu, D., Seide, F., Seltzer, M., Zweig, G., He, X., Williams, J., et al., 2013. Recent advances in deep learning for speech research at microsoft. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing. IEEE, pp. 8604–8608.
- Deng, L., Yu, D., et al., 2014. Deep learning: Methods and applications. *Found. Trends® Signal Process.* 7 (3–4), 197–387.
- Dong, X.L., Rekatsinas, T., 2018. Data integration and machine learning: A natural synergy. In: Proceedings of the 2018 International Conference on Management of Data. pp. 1645–1650.
- Eisenhardt, K.M., 1989. Building theories from case study research. *Acad. Manag. Rev.* 14 (4), 532–550.
- Fahland, D., Van Der Aalst, W.M., 2013. Simplifying discovered process models in a controlled manner. *Inf. Syst.* 38 (4), 585–605.
- Fei-Fei, L., Fergus, R., Perona, P., 2006. One-shot learning of object categories. *IEEE Trans. Pattern Anal. Mach. Intell.* 28 (4), 594–611.
- Gautam, G., Yadav, D., 2014. Sentiment analysis of twitter data using machine learning approaches and semantic analysis. In: 2014 Seventh International Conference on Contemporary Computing. IC3, IEEE, pp. 437–442.
- Gilyazev, R., Turdakov, D.Y., 2018. Active learning and crowdsourcing: A survey of optimization methods for data labeling. *Program. Comput. Softw.* 44 (6), 476–491.
- Goodhope, K., Koshy, J., Kreps, J., Narkhede, N., Park, R., Rao, J., Ye, V.Y., 2012. Building linkedin's real-time activity data pipeline. *IEEE Data Eng. Bull.* 35 (2), 33–45.
- Gruener, R., Cheng, O., Litvin, Y., 2018. Introducing petastorm: Uber ATG's data access library for deep learning | uber engineering blog. <https://eng.uber.com/petastorm/>. (Accessed 05 January 2020).
- Guo, C., Berkjhahn, F., 2016. Entity embeddings of categorical variables. arXiv preprint arXiv:1604.06737.
- Halevy, A., Korn, F., Noy, N.F., Olston, C., Polyzotis, N., Roy, S., Whang, S.E., 2016. Goods: Organizing google's datasets. In: Proceedings of the 2016 International Conference on Management of Data. pp. 795–806.
- He, H., Bai, Y., Garcia, E.A., Li, S., 2008. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In: 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence). IEEE, pp. 1322–1328.
- Hendrycks, D., Mazeika, M., Kadavath, S., Song, D., 2019. Using self-supervised learning can improve model robustness and uncertainty. arXiv preprint arXiv:1906.12340.
- Holmberg, L., Davidsson, P., Linde, P., 2020. A feature space focus in machine teaching. In: 2020 IEEE International Conference on Pervasive Computing and Communications Workshops. PerCom Workshops, IEEE, pp. 1–2.
- Hu, Z., Tan, B., Salakhutdinov, R., Mitchell, T., Xing, E.P., 2019. Learning data manipulation for augmentation and weighting. arXiv preprint arXiv:1910.12795.
- Hynes, N., Sculley, D., Terry, M., 2017. The data linter: Lightweight, automated sanity checking for ml data sets. In: NIPS MLSys Workshop. pp. 1–7.
- Jerez, J.M., Molina, I., García-Laencina, P.J., Alba, E., Ribelles, N., Martín, M., Franco, L., 2010. Missing data imputation using statistical and machine learning methods in a real breast cancer problem. *Artif. Intell. Med.* 50 (2), 105–115.
- Jones, N., 2014. Computer science: The learning machines. *Nat. News* 505 (7482), 146.
- Kahng, M., Andrews, P.Y., Kalro, A., Chau, D.H.P., 2017. A ct v is: Visual exploration of industry-scale deep neural network models. *IEEE Trans. Vis. Comput. Graphics* 24 (1), 88–97.
- Kepuska, V., Bohouta, G., 2018. Next-generation of virtual personal assistants (microsoft cortana, apple siri, amazon alexa and google home). In: 2018 IEEE 8th Annual Computing and Communication Workshop and Conference. CCWC, IEEE, pp. 99–103.
- Kim, W., Choi, B.-J., Hong, E.-K., Kim, S.-K., Lee, D., 2003. A taxonomy of dirty data. *Data Min. Knowl. Discov.* 7 (1), 81–99.
- Kitchenham, B., 2004. Procedures for performing systematic reviews. Keele, UK, Keele University.
- Krawczyk, B., Schaefer, G., Wozniak, M., 2012. Breast thermogram analysis using a cost-sensitive multiple classifier system. In: Proceedings of 2012 IEEE-EMBS International Conference on Biomedical and Health Informatics. IEEE, pp. 507–510.
- Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems. pp. 1097–1105.
- Labrinidis, A., Jagadish, H.V., 2012. Challenges and opportunities with big data. *Proc. VLDB Endow.* 5 (12), 2032–2033.
- Lakshminarayanan, K., Harp, S.A., Goldman, R.P., Samad, T., et al., 1996. Imputation of missing data using machine learning techniques.. In: KDD. pp. 140–145.
- LeCun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. *Nature* 521 (7553), 436–444.
- LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D., 1989. Backpropagation applied to handwritten zip code recognition. *Neural Comput.* 1 (4), 541–551.
- Lin, J., Kolcz, A., 2012. Large-scale machine learning at twitter. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. pp. 793–804.
- Litjens, G., Kooi, T., Bejnordi, B.E., Setio, A.A.A., Ciompi, F., Ghafoorian, M., Van Der Laak, J.A., Van Ginneken, B., Sánchez, C.I., 2017. A survey on deep learning in medical image analysis. *Med. Image Anal.* 42, 60–88.
- Lwakatare, L.E., Ränge, E., Crnkovic, I., Bosch, J., 2021. On the experiences of adopting automated data validation in an industrial machine learning project. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice. ICSE-SEIP, IEEE, pp. 248–257.
- Maguire, M., Delahunt, B., 2017. Doing a thematic analysis: A practical, step-by-step guide for learning and teaching scholars.. *All Ireland J. Higher Educ.* 9 (3).
- Malone, T.W., Laubacher, R., Dellarocas, C., 2009. Harnessing crowds: Mapping the genome of collective intelligence.
- Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., Hung Byers, A., et al., 2011. Big Data: The Next Frontier for Innovation, Competition, and Productivity. McKinsey Global Institute.
- Matheus, R., Janssen, M., Maheshwari, D., 2018. Data science empowering the public: Data-driven dashboards for transparent and accountable decision-making in smart cities. *Gov. Inf. Q.* 101284.
- Meredith, J., 1998. Building operations management theory through case and field research. *J. Oper. Manage.* 16 (4), 441–454.
- Misra, I., Maaten, L.v.d., 2020. Self-supervised learning of pretext-invariant representations. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 6707–6717.
- Munappy, A., Bosch, J., Olsson, H.H., Arpteg, A., Brinne, B., 2019. Data management challenges for deep learning. In: 2019 45th Euromicro Conference on Software Engineering and Advanced Applications. SEAA, IEEE, pp. 140–147.
- Munappy, A.R., Mattos, D.I., Bosch, J., Olsson, H.H., Dakkak, A., 2020. From ad-hoc data analytics to dataops. In: Proceedings of the International Conference on Software and System Processes. pp. 165–174.
- Najafabadi, M.M., Villanustre, F., Khoshgoftaar, T.M., Seliya, N., Wald, R., Muhameragic, E., 2015. Deep learning applications and challenges in big data analytics. *J. Big Data* 2 (1), 1.
- National Security Agency, 2013. The National Security Agency: Missions, Authorities, Oversight and Partnerships. National Security Agency Ft. Meade, MD.
- Nguyen, A., Wallace, B., Lease, M., 2015. Combining crowd and expert labels using decision theoretic active learning. In: Proceedings of the AAAI Conference on Human Computation and Crowdsourcing, Vol. 3, no. 1.
- Nowak, S., Rüger, S., 2010. How reliable are annotations via crowdsourcing: A study about inter-annotator agreement for multi-label image annotation. In: Proceedings of the International Conference on Multimedia Information Retrieval. pp. 557–566.
- Oguntimilehin, A., Ademola, E., 2014. A review of big data management, benefits and challenges. *Rev. Big Data Manag., Benefits Chall.* 5 (6), 1–7.
- Raj, A., Bosch, J., Olsson, H.H., Wang, T.J., 2020. Modelling data pipelines. In: 2020 46th Euromicro Conference on Software Engineering and Advanced Applications. SEAA, IEEE, pp. 13–20.
- Ranzato, M., Boureau, Y.-L., LeCun, Y., et al., 2007. Sparse feature learning for deep belief networks. *Adv. Neural Inf. Process. Syst.* 20, 1185–1192.
- Rao, Q., Frtunekj, J., 2018. Deep learning for self-driving cars: Chances and challenges. In: Proceedings of the 1st International Workshop on Software Engineering for AI in Autonomous Systems. pp. 35–38.

- Rastogi, R., 2018. Machine learning@ amazon. In: The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval. pp. 1337–1338.
- Romera-Paredes, B., Torr, P., 2015. An embarrassingly simple approach to zero-shot learning. In: International Conference on Machine Learning. PMLR, pp. 2152–2161.
- Roy, A., Sun, J., Mahoney, R., Alonzi, L., Adams, S., Beling, P., 2018. Deep learning detecting fraud in credit card transactions. In: 2018 Systems and Information Engineering Design Symposium. SIEDS, IEEE, pp. 129–134.
- Runeson, P., Höst, M., 2009. Guidelines for conducting and reporting case study research in software engineering. Empir. Softw. Eng. 14 (2), 131–164.
- Sergeev, A., Del Balso, M., 2018. Horovod: Fast and easy distributed deep learning in TensorFlow. arXiv preprint arXiv:1802.05799.
- Settles, B., 2009. Active learning literature survey.
- Settles, B., Craven, M., Friedland, L., 2008. Active learning with real annotation costs. In: Proceedings of the NIPS Workshop on Cost-Sensitive Learning, Vol. 1. Vancouver, CA.
- Shi, X., Fan, W., Ren, J., 2008. Actively transfer domain knowledge. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, pp. 342–357.
- Snell, J., Swersky, K., Zemel, R.S., 2017. Prototypical networks for few-shot learning. arXiv preprint arXiv:1703.05175.
- Socher, R., Ganjoo, M., Sridhar, H., Bastani, O., Manning, C.D., Ng, A.Y., 2013. Zero-shot learning through cross-modal transfer. arXiv preprint arXiv:1301.3666.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., 2014. Dropout: A simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. 15 (1), 1929–1958.
- Stadler, J.G., Donlon, K., Siewert, J.D., Franken, T., Lewis, N.E., 2016. Improving the efficiency and ease of healthcare analysis through use of data visualization dashboards. Big Data 4 (2), 129–135.
- Stuart, I., McCutcheon, D., Handfield, R., McLachlin, R., Samson, D., 2002. Effective case research in operations management: A process perspective. J. Oper. Manage. 20 (5), 419–433.
- Sucholutsky, I., Schonlau, M., 2020. 'Less than one'-shot learning: Learning n classes from m < n samples. arXiv preprint arXiv:2009.08449.
- Sun, H., Hu, S., McIntosh, S., Cao, Y., 2018. Big data trip classification on the new york city taxi and uber sensor network. J. Internet Technol. 19 (2), 591–598.
- Tanzil, S.M.S., Hoiles, W., Krishnamurthy, V., 2017. Adaptive scheme for caching YouTube content in a cellular network: Machine learning approach. IEEE Access 5, 5870–5881.
- Tole, A.A., et al., 2013. Big data challenges. Database Syst. J. 4 (3), 31–40.
- Triantafillou, E., Zemel, R., Urtasun, R., 2017. Few-shot learning through an information retrieval lens. arXiv preprint arXiv:1707.02610.
- Triguero, I., García, S., Herrera, F., 2015. Self-labeled techniques for semi-supervised learning: Taxonomy, software and empirical study. Knowl. Inf. Syst. 42 (2), 245–284.
- Tur, G., De Mori, R., 2011. Spoken Language Understanding: Systems for Extracting Semantic Information from Speech. John Wiley & Sons.
- Van Alstyne, M.W., Parker, G.G., Choudary, S.P., 2016. Pipelines, platforms, and the new rules of strategy. Harv. Bus. Rev. 94 (4), 54–62.
- Verner, J.M., Sampson, J., Tasic, V., Bakar, N.A., Kitchenham, B.A., 2009. Guidelines for industrially-based multiple case studies in software engineering. In: 2009 Third International Conference on Research Challenges in Information Science. IEEE, pp. 313–324.
- Viaene, S., 2013. Data scientists aren't domain experts. IT Prof. 15 (6), 12–17.
- Wang, W., Zhang, M., Chen, G., Jagadish, H., Ooi, B.C., Tan, K.-L., 2016. Database meets deep learning: Challenges and opportunities. ACM SIGMOD Rec. 45 (2), 17–22.
- Wen, Q., Sun, L., Yang, F., Song, X., Gao, J., Wang, X., Xu, H., 2020. Time series data augmentation for deep learning: A survey. arXiv preprint arXiv:2002.12478.
- Whang, S.E., Lee, J.-G., 2020. Data collection and quality challenges for deep learning. Proc. VLDB Endow. 13 (12), 3429–3432.
- Whittaker, C., Ryner, B., Nazif, M., 2010. Large-scale automatic classification of phishing pages. NDSS '10.
- Xian, Y., Schiele, B., Akata, Z., 2017. Zero-shot learning—the good, the bad and the ugly. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4582–4591.
- Yin, R.K., 2003. Case study research design and methods third edition. Appl. Soc. Res. Methods Ser. 5.
- Yin, R.K., 2013. Validity and generalization in future case study evaluations. Evaluation 19 (3), 321–332.
- Zhai, X., Oliver, A., Kolesnikov, A., Beyer, L., 2019. S4I: Self-supervised semi-supervised learning. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 1476–1485.
- Zhang, A., Ballas, N., Pineau, J., 2018. A dissection of overfitting and generalization in continuous reinforcement learning. arXiv preprint arXiv:1806.07937.
- Zhang, C., Kumar, A., Ré, C., 2016. Materialization optimizations for feature selection workloads. ACM Trans. Database Syst. 41 (1), 1–32.
- Zhang, S., Yao, L., Sun, A., Tay, Y., 2019. Deep learning based recommender system: A survey and new perspectives. ACM Comput. Surv. 52 (1), 1–38.
- Zhang, S., Zhang, C., Yang, Q., 2003. Data preparation for data mining. Appl. Artif. Intell. 17 (5–6), 375–381.
- Zhou, Z.-H., Zhan, D.-C., Yang, Q., 2007. Semi-supervised learning with very few labeled training examples. In: AAAI, Vol. 675680.
- Zhu, X., Goldberg, A.B., 2009. Introduction to semi-supervised learning. Synth. Lect. Artif. Intell. Mach. Learn. 3 (1), 1–130.
- Ziv, J., Lempel, A., 1977. A universal algorithm for sequential data compression. IEEE Trans. Inform. Theory 23 (3), 337–343.

Aiswarya Raj Munappy received the Licentiate degree in software engineering from the Chalmers University of Technology, Sweden, in 2021. She currently works as a Ph.D. student at the Chalmers University of Technology, Department of Computer Science. Her current research interests include the AI engineering, data management, deep learning, data pipelines and DataOps.

Jan Bosch is a Professor in the Software Engineering at the [UniversityofGroningen](#) and at [ChalmersUniversityofTechnology](#). He received his M.Sc. in computer science in 1991 from the [UniversityofTwente](#), and in 1995 his Ph.D. degree in computer science from [LundUniversity](#). He has pioneered research in software architecture, digitalization and AI Engineering, and has published more than 500 research articles. Jan Bosch is the director of Software Center organization which aim to develop methods and tools for enhancing the productivity of collaborating organizations in various development phases.

Helena Holmström Olsson is a professor at the Department of Computer Science and Media Technology at Malmö University, Sweden. She received her Ph.D. from University of Gothenburg in 2004. She is a senior researcher in Software Center. Her research interest is in Software Engineering focused on AI engineering, data-driven development, software and business ecosystems, business agility and the overall digital transformation of the embedded systems domain.

Anders Arpteg has been working with AI for 20 years in both academia and industry and holds a Ph.D. in AI from Linköping University. Previously, he headed up a research group at Spotify and has headed up the research team at Peltarion. Anders is a member of the AI Innovation of Sweden steering committee, an AI adviser for the Swedish government, a member of the Swedish AI Agenda, Chairman of the Machine Learning Stockholm meetup group and a member of several other advisory boards.

Björn Brinne is an experienced Data Science leader with a demonstrated history of working in the computer software industry. He is skilled in data science, machine learning, and analytics. He is a strong engineering professional with a PhD in theoretical physics from Stockholm University and has contributed to many research papers across a range of academic fields, including computer science, string theory and computational biology. Björn Brinne has over a decade of experience working in data science at companies such as Truecaller, King and Electronic Arts.



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

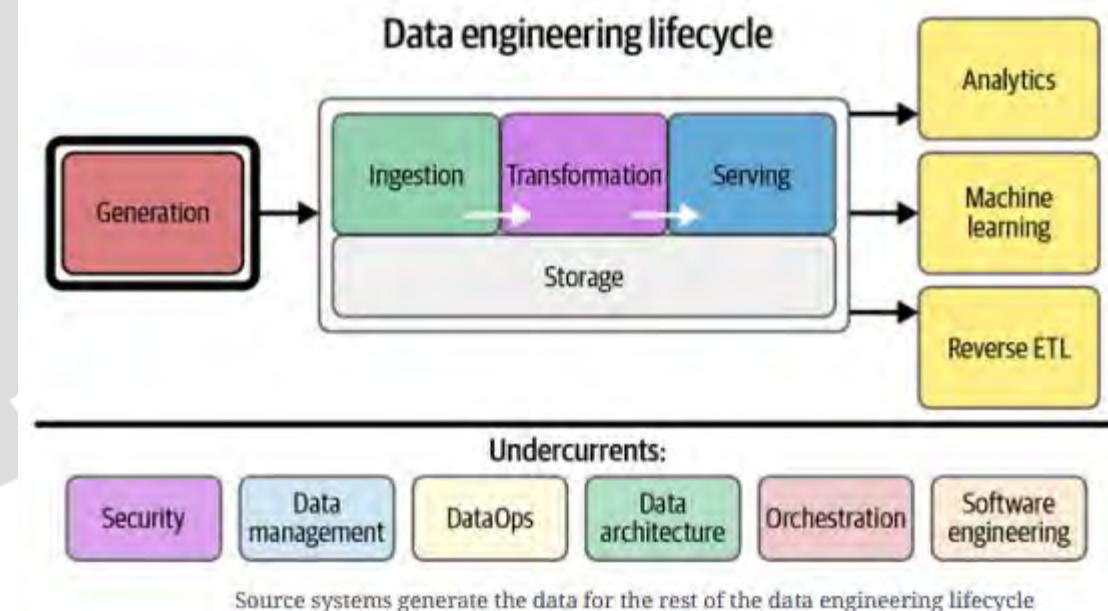
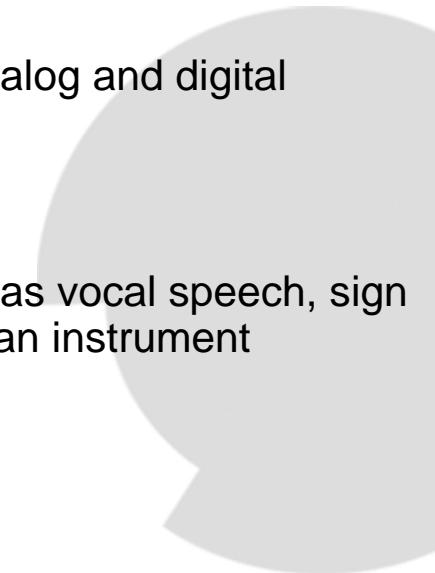
Data Generation in Source Systems

Pravin Y Pawar

Adapted from
Fundamentals of Data Engineering
By Joe Reis and Matt Housley

Sources of Data: How Is Data Created?

- Essential to understand how data is created, as
 - Data is an unorganized, context-less collection of facts and figures
 - Can be created in many ways, both analog and digital
- Analog data
 - creation occurs in the real world, such as vocal speech, sign language, writing on paper, or playing an instrument
 - often transient
- Digital data
 - either created by converting analog data to digital form or is the native product of a digital system
 - A mobile texting app that converts analog speech into digital text
 - A credit card transaction on an ecommerce platform



Source Systems

Relational databases

- Most common application backends - developed at IBM in the 1970s and popularized by Oracle in the 1980s
 - growth of the internet saw the rise of the LAMP stack (Linux, Apache web server, MySQL, PHP) and an explosion of vendor and open source RDBMS options
- Data is stored in a table of relations (**rows**), and each relation contains multiple fields (**columns**)
- Each relation in the table has the **same schema**
- Rows are typically stored as a **contiguous sequence of bytes on disk**
- Tables are typically **indexed** by a **primary key**, a unique field for each row in the table
 - The indexing strategy for the primary key is closely connected with the layout of the table on disk
- Tables can also have **various foreign keys** —fields with values connected with the values of primary keys in other tables,
 - facilitating joins, and allowing for complex schemas that spread data across multiple tables
- **Normalization** is a strategy for ensuring that data in records is not duplicated in multiple places
 - avoiding the need to update states in multiple locations at once and preventing inconsistencies
- RDBMS systems are typically **ACID compliant**
- Relational database systems ideal for storing rapidly changing application states
 - Combining a normalized schema, ACID compliance, and support for high transaction rates
 - Challenge is to determine how to capture state information over time

Source Systems(2)

Non-relational databases: NoSQL

- NoSQL, not only SQL, refers to a whole class of databases that abandon the relational paradigm
 - widely adopted to describe a universe of “new school” databases, alternatives to relational databases
 - dropping relational constraints can improve performance, scalability, and schema flexibility
 - abandon various RDBMS characteristics, such as strong consistency, joins, or a fixed schema
- Numerous flavors of NoSQL database designed for almost any imaginable use case
 - key-value
 - document
 - wide-column
 - graph
 - search
 - time series
- all wildly popular and enjoy widespread adoption
- Should understand
 - these types of databases
 - including usage considerations
 - the structure of the data they store
 - and how to leverage each in the data engineering lifecycle

Source Systems(3)

Key-Value stores

- A non-relational database that retrieves records using a key that uniquely identifies each record
 - similar to hash map or dictionary data structures presented in many programming languages but potentially more scalable
 - encompass several NoSQL database types —document stores and wide column databases
- Different types of key-value databases offer a variety of performance characteristics to serve various application needs
- In-memory key-value databases are popular for caching session data for web and mobile applications, where ultra-fast lookup and high concurrency are required
 - Storage in these systems is typically temporary - if the database shuts down, the data disappears
 - Such caches can reduce pressure on the main application database and serve speedy responses
- Key-value stores can also serve applications requiring high-durability persistence
 - An ecommerce application may need to save and update massive amounts of event state changes for a user and their orders
 - A user logs into the ecommerce application, clicks around various screens, adds items to a shopping cart, and then checks out
 - Each event must be durably stored for retrieval
 - Key-value stores often persist data to disk and across multiple nodes to support such use cases

Source Systems(4)

Document stores

- A specialized key-value store
 - A document is a nested object - each document as a JSON object - are stored in collections and retrieved by key
 - A collection is roughly equivalent to a table in a relational database
- Does not support joins - data cannot be easily normalized, i.e., split across multiple tables
 - Ideally, all related data can be stored in the same document.
 - In many cases, the same data must be stored in multiple documents spread across numerous collections
- Generally embrace all the flexibility of JSON and don't enforce schema or types - blessing and a curse
 - allows the schema to be highly flexible and expressive - can also evolve as an application grows
 - become absolute nightmares to manage and query - data may become inconsistent and bloated over time
- Support the creation of indexes and lookup tables to allow retrieval of documents by specific properties
 - invaluable in application development when you need to search for documents in various ways
- Generally not ACID compliant - many are eventually consistent
 - Allowing data distribution across a cluster is a boon for scaling and performance but can lead to catastrophes when engineers and developers don't understand the implications
- To run analytics on document stores, engineers generally must run a full scan to extract all data from a collection
 - can have both performance and cost implications

Source Systems(5)

Wide-column database

- Optimized for storing massive amounts of data with high transaction rates and extremely low latency
- Can scale to extremely high write rates and vast amounts of data
 - petabytes of data
 - millions of requests per second
 - and sub10ms latency
- have made wide-column databases popular in ecommerce, fintech, ad tech, IoT, and real-time personalization applications
- Support rapid scans of massive amounts of data, but do not support complex queries
 - have only a single index (the row key) for lookups
 - must generally extract data and send it to a secondary analytics system to run complex queries to deal with these limitations
 - can be accomplished by running large scans for the extraction or employing CDC to capture an event stream

Source Systems(6)

Search and Time series databases

- Search
 - Non-relational database used to search data's complex and straightforward semantic and structural characteristics
 - Search databases are popular for fast search and retrieval and can be found in various applications
 - Elasticsearch, Apache Solr or Lucene, or Algolia
 - Two prominent use cases exist for a search database: text search and log analysis
 - Text search involves searching a body of text for keywords or phrases, matching on exact, fuzzy, or semantically similar matches
 - Log analysis is typically used for anomaly detection, real-time monitoring, security analytics, and operational analytics
- Time series
 - A series of values organized by time - stock prices might move as trades are executed throughout the day
 - Any events that are recorded over time — either regularly or sporadically — are time-series data
 - Optimized for retrieving and statistical processing of time-series data
 - Address the needs of growing, high-velocity data volumes from IoT, event and application logs, ad tech, and fintech
 - Often utilize memory buffering to support fast writes and reads
 - Measurement data is generated regularly, such as temperature or air-quality sensors
 - Event-based data is irregular and created every time an event occurs — for instance, when a motion sensor detects movement
- The schema for a time series typically contains a timestamp and a small set of fields
 - makes time-series databases suitable for operational analytics but not great for BI use cases
 - Joins are not common, though some quasi time-series databases such as Apache Druid support joins

Source Systems(7)

APIs - REST

- APIs are now a standard and pervasive way of exchanging data in the cloud, for SaaS platforms, and between internal company systems
- REST
 - Currently the dominant API paradigm - REpresentational State Transfer
 - set of practices and philosophies for building HTTP web APIs was laid out by Roy Fielding in 2000 in a PhD dissertation
 - built around HTTP verbs, such as GET and PUT; in practice, modern REST uses only a handful of the verb mappings outlined in the original dissertation.
 - Interactions are stateless - each REST call is independent
 - can change the system's state, but these changes are global, applying to the full system rather than a current session
 - Great variation in levels of API abstraction
 - APIs are merely a thin wrapper over internals that provides the minimum functionality required to protect the system from user requests
 - API is a masterpiece of engineering that prepares data for analytics applications and supports advanced reporting
 - Data providers frequently supply client libraries in various languages, especially in Python
 - remove much of the boilerplate labor of building API interaction code
 - handle critical details such as authentication and map fundamental methods into accessible classes
 - Various services and open source libraries have emerged to interact with APIs and manage data synchronization
 - Many SaaS and open source vendors provide off-the-shelf connectors for common APIs
 - Platforms also simplify the process of building custom connectors as required
 - Numerous data APIs without client libraries or out-of-the-box connector support

Source Systems(8)

Other API systems

- GraphQL
 - Created at Facebook as a query language for application data and an alternative to generic REST APIs
 - REST APIs generally restrict queries to a specific data model, GraphQL opens up the possibility of retrieving multiple data models in a single request
 - allows for more flexible and expressive queries than with REST
 - built around JSON and returns data in a shape resembling the JSON query
- Webhooks
 - A simple event-based data-transmission pattern - aka reverse APIs
 - Data source can be an application backend, a web page, or a mobile app
 - When specified events happen in the source system, this triggers a call to an HTTP endpoint hosted by the data consumer
 - Endpoint can do various things with the POST event data, potentially triggering a downstream process or storing the data for future use
- RPC and gRPC
 - A remote procedure call (RPC) is commonly used in distributed computing - allows to run a procedure on a remote system
 - gRPC is a remote procedure call library developed internally at Google in 2015 and later released as an open standard
 - Many Google services, such as Google Ads and GCP, offer gRPC APIs. gRPC is built around the Protocol Buffers open data serialization standard
 - gRPC emphasizes the efficient bidirectional exchange of data over HTTP/2.
 - Efficiency refers to aspects such as CPU utilization, power consumption, battery life, and bandwidth
 - imposes much more specific technical standards than REST
 - Allows the use of common client libraries and allowing engineers to develop a skill set that will apply to any gRPC interaction code.

Source Systems(9)

Data Sharing

- The core concept of cloud data sharing is that a multitenant system supports security policies for sharing data among tenants
 - Any public cloud object storage system with a fine-grained permission system can be a platform for data sharing
 - Popular cloud data-warehouse platforms also support data-sharing capabilities
 - Data can also be shared through download or exchange over email, but a multitenant system makes the process much easier
- Many modern sharing platforms (especially cloud data warehouses) support row, column, and sensitive data filtering
 - streamlines the notion of the data marketplace, available on several popular clouds and data platforms
 - Data marketplaces provide a centralized location for data commerce
 - where data providers can advertise their offerings and sell them without worrying about the details of managing network access to data systems
- Data sharing can also streamline data pipelines within an organization
 - allows units of an organization to manage their data and selectively share it with other units
 - Allows individual units to manage their compute and query costs separately, facilitating data decentralization
 - facilitates decentralized data management patterns such as data mesh

Source Systems(10)

Third-Party Data Sources

- Every company is essentially now a technology company
 - consequence is want to make their data available to their customers and users, either as part of their service or as a separate subscription
- For example,
 - US Bureau of Labor Statistics publishes various statistics about the US labor market
 - NASA publishes various data from its research initiatives
 - Facebook shares data with businesses that advertise on its platform
- Why would companies want to make their data available?
 - Data is sticky, and a flywheel is created by allowing users to integrate and extend their application into a user's application
 - Greater user adoption and usage means more data, which means users can integrate more data into their applications and data systems
 - Side effect is there are now almost infinite sources of third-party data
- Direct third-party data access is commonly done via APIs, through data sharing on a cloud platform, or through data download
 - APIs often provide deep integration capabilities, allowing customers to pull and push data
- For example,
 - Many CRMs offer APIs that their users can integrate into their systems and applications
 - A common workflow to get data from a CRM, blend the CRM data through the customer scoring model, and then use reverse ETL to send that data back into
 - CRM for salespeople to contact better-qualified leads

Source Systems(11)

Message Queues

- Event-driven architectures are pervasive in software applications and are poised to grow their popularity even further
 - Message queues and event-streaming platforms—critical layers in event-driven architectures—are easier to set up and manage in a cloud environment
 - Rise of data apps—applications that directly integrate real-time analytics—are growing from strength to strength
- As source systems, message queues and event-streaming platforms are used in numerous ways
 - from routing messages between microservices ingesting millions of events per second of event data from web, mobile, and IoT applications
- Message queues
 - A mechanism to asynchronously send data (usually as small individual messages, in the kilobytes) between discrete systems using a publish and subscribe model
 - Data is published to a message queue and is delivered to one or more subscribers
 - Subscriber acknowledges receipt of the message, removing it from the queue
- Message queues
 - allow applications and systems to be decoupled from each other and are widely used in microservices architectures
 - buffers messages to handle transient load spikes and makes messages durable through a distributed architecture with replication
 - critical ingredient for decoupled microservices and event-driven architectures

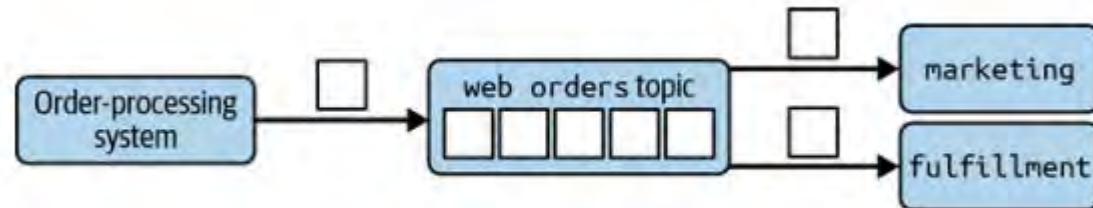
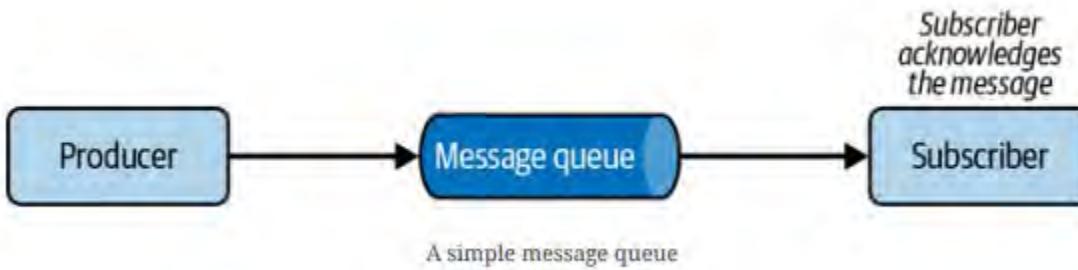
Source Systems(12)

Event-Streaming Platforms

- Is a continuation of a message queue in that messages are passed from producers to consumers
- Big difference between messages and streams is that
 - A message queue is primarily used to route messages with certain delivery guarantees
 - An event-streaming platform is used to ingest and process data in an ordered log of records
- In an event-streaming platform, data is retained for a while, and it is possible to replay messages from a past point in time.
- Topics
 - A producer streams events to a topic, a collection of related events
 - might contain fraud alerts, customer orders, or temperature readings from IoT devices
 - can have zero, one, or multiple producers and consumers on most event-streaming platforms
- Stream partitions
 - are subdivisions of a stream into multiple streams
 - A good analogy is a multilane freeway - Having multiple lanes allows for parallelism and higher throughput
 - Messages are distributed across partitions by partition key
 - Messages with the same partition key will always end up in the same partition

Source Systems(13)

Message Queues and Event-Streaming Platforms





Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

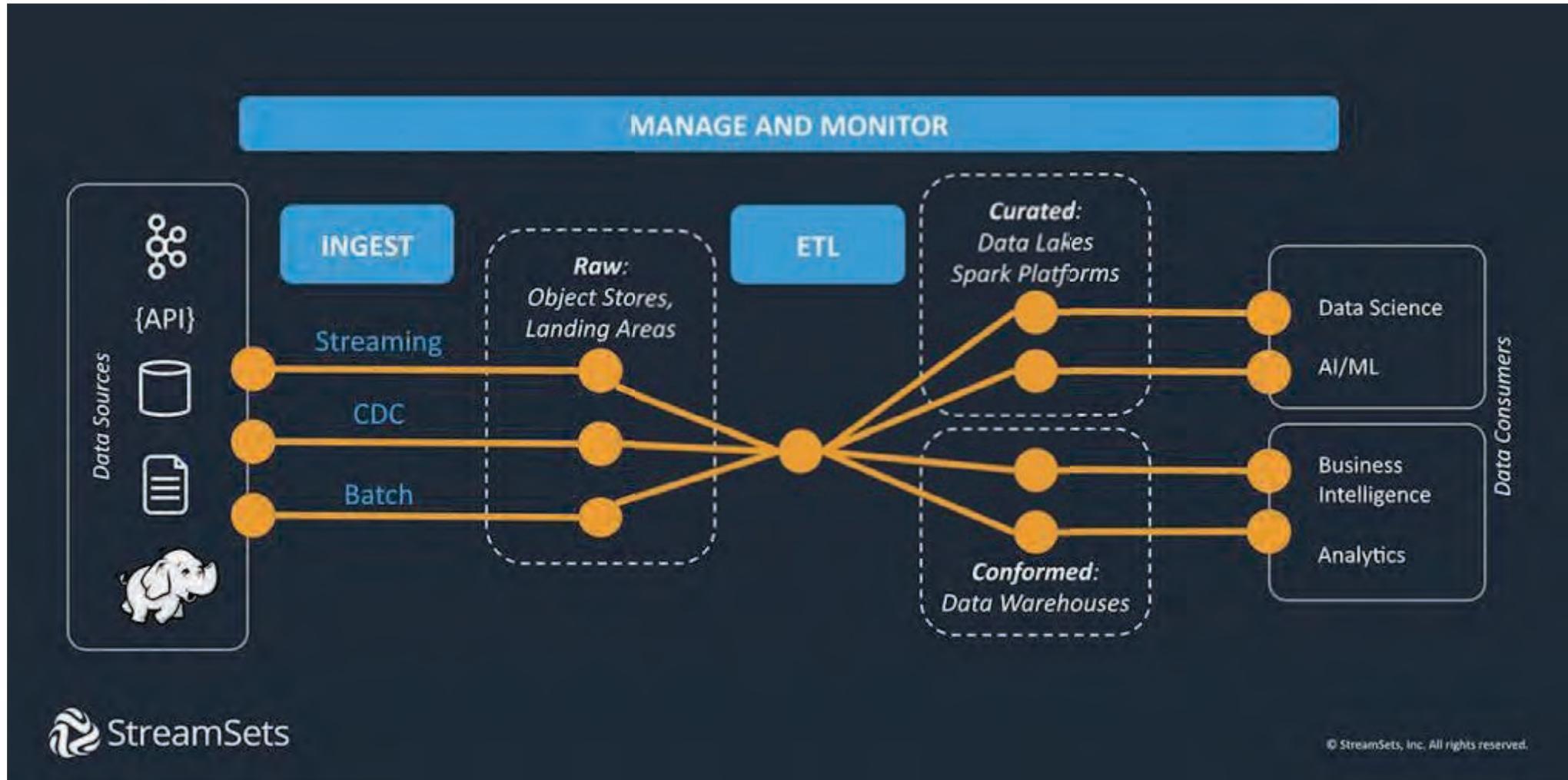
Data Ingestion

Pravin Y Pawar

Data Ingestion

- Process of moving data from a source into a landing area or an object store where it can be used for ad hoc queries and analytics.
 - A simple data ingestion pipeline consumes data from a point of origin, cleans it up a bit, then writes it to a destination
- Importance
 - Helps teams go fast!
 - Scope of any given data pipeline **is deliberately narrow**, giving data teams flexibility and agility at scale
 - Once parameters are set, **data analysts and data scientists** can easily build a single data pipeline to **move data to their system of choice**
- Common examples of data ingestion include:
 - Move data from Salesforce.com to a data warehouse then analyze with Tableau
 - Capture data from a Twitter feed for real-time sentiment analysis
 - Acquire data for training machine learning models and experimentation
- Modern Data Integration Begins with Data Ingestion
 - Data engineers use data ingestion pipelines to better **handle the scale and complexity of business demands for data**
 - Lots of intent-driven data pipelines operating continuously across the organization without direct involvement of a development team
- **Data ingestion has become a key component of self-service platforms for analysts and data scientists to access data for real-time analytics, machine learning and AI workloads.**

Working



Working(2)

- Data ingestion **extracts data from the source** where it was created or originally stored
 - Then **loads data into a destination or staging area**
- A simple data ingestion pipeline
 - might apply **one or more light transformations**
 - enriching or filtering data
 - before writing it to some set of destinations, a data store or a message queue
- More complex transformations
 - such as **joins, aggregates, and sorts**
 - for specific analytics, applications and reporting systems
 - can be done with **additional pipelines**

Components

- Data Sources
 - Data teams have moved way beyond the walled garden of the **enterprise data center** for insight
 - increasingly load data from across **business units as well as 3rd party and unstructured** data, start data loads, when and where they need it
 - Common data source types:
 - Apache Kafka
 - JDBC
 - Oracle CDC
 - HTTP Clients
 - HDFS
- Data Destinations
 - A data ingestion pipeline may simply send data to an **application or messaging system**, or store ingested data in a **data lake or cloud object storage** for use in relational and **NoSQL databases or data warehouses**
 - Common destination:
 - Apache Kafka
 - JDBC
 - Snowflake
 - Amazon S3
 - Databricks
- Cloud Data Migration
 - Enterprise business processes are **moving to cloud-based platforms for storing, processing and applications**
 - Data ingestion workloads have become essential **to cloud migration**
 - But moving data out of silos and into agile cloud data lakes or powerful cloud data warehouses, generates **some uncomfortable questions**
- The more data platforms can automate and operationalize the what-ifs of data ingestion, the better they can support the growing demand for continuous, reliable data.

Data Ingestion vs Data Integration

- Data ingestion originated as a small part of data integration, a more complex process required to make data consumable in new systems before loading it
- Data integration usually requires advance specification from source to schema to transformation to destination
- With data ingestion, a few light transformations may be made, such as masking personally identifiable information (PII)
 - but most of the work depends on the end use and happens after landing the data
- Think of it this way:
 - Data integration includes processes to prepare data to be consumable at its final destination
 - Data ingestion gets data to places where preparation happens in response to downstream needs
- Data ingestion works well for streaming data that can be used immediately with very few transformations or as a way to collect data for ad hoc analysis
 - By focusing on the ingestion part of the data lifecycle, companies have been able to speed up the availability of data for innovation and growth

Data Ingestion Challenges

- With the rise of **big data, cloud computing, and the demand for real-time analytics**, the capacity for data increased dramatically
 - old **ETL** processes began to **slow data teams** down compared to the ELT model
- Complexity Takes Time**
 - The to-do list for data engineering is long and getting **longer**
 - Building data pipelines from scratch every time a new source or business need comes up **slows down the whole data team**
- Change Takes Time**
 - Every **change** or evolution of a target system **generates 10-20 hours of work for a data engineer**
 - 90% of time will be spent on **maintenance and break-fix, changes** that are considered data drift
- Maintenance and Rework Takes Time**
 - Doing the **same thing over and over** with a lot of troubleshooting and debugging doesn't leave much time for **innovation or ramping** up on new technologies

Types of Data Ingestion Tools

- Hand Coding
 - One way to ingest data may be to hand code a data pipeline, assuming know how to code and are familiar with the **languages** needed
 - **gives the greatest control**
 - may spend a lot of time **working and reworking code**
- Single-purpose Tools
 - Basic data ingestion tools provide a **drag-and-drop interface** with lots of pre-built connectors and transformation
 - a **quick way** to get a lot done or to enable less skilled data consumers,
 - **management** of many drag-and-drop data pipelines will be **cumbersome**
 - **can't share work** with team or the analysts and data scientists knocking on door
- Data Integration Platforms
 - Traditional data integration platforms **incorporate features** for every step of the data value chain
 - means most likely **need developers** and architectures specific to each domain
 - making it **difficult to move fast** and adapt easily to change
- A DataOps Approach
 - Applying **agile methodologies** to **data**, a DataOps approach to data pipelines **automates as much as possible** and **abstracts away the "how" of implementation**
 - Data engineers can focus on the **"what"** of the **data**, and responding to business needs

Key Engineering Considerations for the Ingestion Phase

Primary considerations related to data ingestion

- What's the use case for the data is ingesting?
- Can this data be reused and avoid ingesting multiple versions of the same dataset?
- Where is the data going? What's the destination?
- How often should the data be updated from the source?
- What is the expected data volume?
- What format is the data in? Can downstream storage and transformation accept this format?
- Is the source data in good shape for immediate downstream use? Is the data of good quality?
- What post-processing is required to serve it? What are data-quality risks?
- Does the data require in-flight processing for downstream ingestion if the data is from a streaming source?

Reference

Data Ingestion: Tools, Types, and Key Concepts

How to get data from where it starts to where it can make a difference

<https://streamsets.com/learn/data-ingestion/>



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

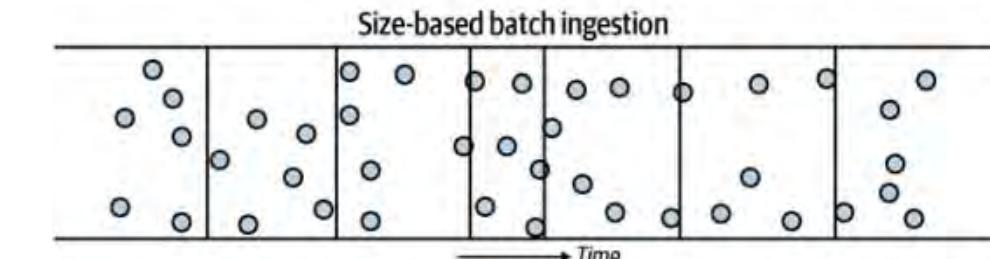
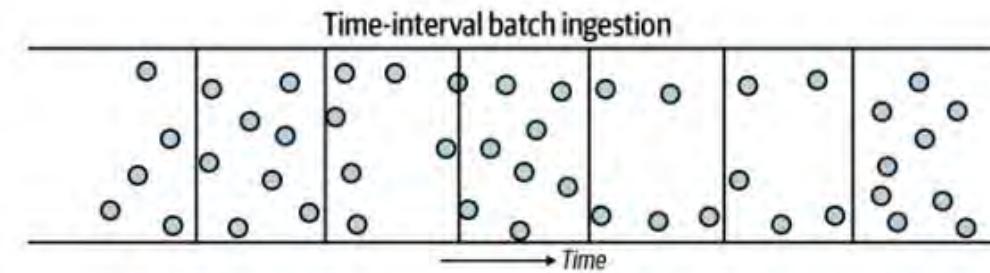
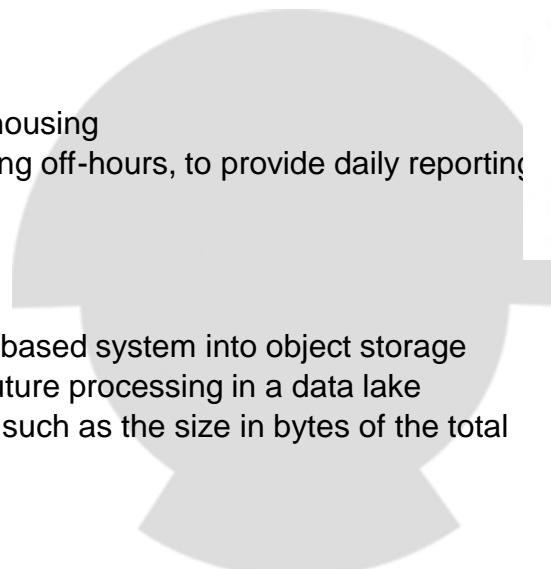
Data Ingestion Considerations

Pravin Y Pawar

Adapted from
Fundamentals of Data Engineering
By Joe Reis and Matt Housley

Batch Ingestion

- Batch ingestion - processing data in bulk, is often a convenient way to ingest data
 - data is ingested by taking a subset of data from a source system, based either on a time interval or the size of accumulated data
- Time-interval batch ingestion
 - widespread in traditional business ETL for data warehousing
 - often used to process data once a day, overnight during off-hours, to provide daily reporting, but other frequencies can also be used
- Size-based batch ingestion
 - quite common when data is moved from a streaming-based system into object storage
 - ultimately, must cut the data into discrete blocks for future processing in a data lake
 - can break data into objects based on various criteria, such as the size in bytes of the total number of events
- Commonly used batch ingestion patterns:
 - Snapshot or differential extraction
 - File-based export and ingestion
 - ETL versus ELT
 - Inserts, updates, and batch size
 - Data migration



Batch Ingestion Considerations

Snapshot or Differential Extraction

- Must choose whether to capture full snapshots of a source system or differential (incremental) updates
 - With full snapshots, grab the entire current state of the source system on each update read
 - With the differential update pattern, can pull only the updates and changes since the last read from the source system
 - Differential updates are ideal for minimizing network traffic and target storage usage
 - Full snapshot reads remain extremely common because of their simplicity

Batch Ingestion Considerations(2)

File-Based Export and Ingestion

- Data is quite often moved between databases and systems using files
 - Data is serialized into files in an exchangeable format, and these files are provided to an ingestion system
- A push-based ingestion pattern
 - data export and preparation work is done on the source system side
- File-based ingestion has several potential advantages over a direct database connection approach
 - often undesirable to allow direct access to backend systems for security reason
 - files can be provided to the target system in various ways
 - Common file-exchange methods are object storage, secure file transfer protocol (SFTP), electronic data interchange (EDI), or secure copy (SCP)

Batch Ingestion Considerations(3)

ETL Versus ELT

- Extremely common ingestion, storage, and transformation patterns encountered in batch workloads
- Extract
 - Getting data from a source system
 - Can be pulling data, or can be push based
 - Extraction may also require reading metadata and schema changes
- Load
 - Once data is extracted,
 - it can either be transformed (ETL) before loading it into a storage destination
 - or simply loaded into storage for future transformation (ELT)
 - should be mindful of the type of system you're loading, the schema of the data, and the performance impact of loading

Batch Ingestion Considerations(4)

Inserts, Updates, and Batch Size

- Batch-oriented systems often perform poorly when users attempt to perform many small-batch operations rather than a smaller number of large operations!
 - common to insert one row at a time in a transactional database, this is a bad pattern for many columnar databases
- Running many small in-place update operations is an even bigger problem
 - causes the database to scan each existing column file to run the update
- Need to understand the appropriate update patterns for the database or data store working with
- Certain technologies are purpose-built for high insert rates
 - Apache Druid and Apache Pinot can handle high insert rates
 - SingleStore can manage hybrid workloads that combine OLAP and OLTP characteristics
 - BigQuery performs poorly on a high rate of vanilla SQL single-row inserts but extremely well if data is fed in through its stream buffer

Batch Ingestion Considerations(5)

Data Migration

- Migrating data to a new database or environment is **not usually trivial**, and data needs to be moved in bulk
 - moving data sizes that are hundreds of terabytes or much larger, often involving the migration of specific tables and moving entire databases and systems
- As is often the case for data ingestion, **schema management** is a crucial consideration
 - No matter how closely the two databases resemble each other, subtle differences almost always exist in the way they handle schema
 - generally easy to test ingestion of a sample of data and find schema issues before undertaking a complete table migration
- Most data systems **perform best when data is moved in bulk rather than as individual rows or events**
 - File or object storage is often an excellent intermediate stage for transferring data
- Many tools are available **to automate various types of data migrations**
 - Especially for large and complex migrations, look at these options before doing this manually or writing own migration solution

Message and Stream Ingestion Considerations

Schema Evolution and Late-Arriving Data

- Schema Evolution
 - common when handling event data - **fields may be added or removed, or value types might change**
 - can have **unintended impacts** on data pipelines and destinations - potentially impact downstream capabilities
- To alleviate issues
 - if event-processing framework has a **schema registry**, use it to version schema changes
 - **A dead-letter queue** can help investigate issues with events that are not properly handled
 - **Low-fidelity route** (and the most effective) is regularly communicating with upstream stakeholders about potential schema changes and proactively addressing schema changes with the teams introducing these changes instead of reacting to the receiving end of breaking changes
- Late-Arriving Data
 - Event data might arrive late - common when ingesting data - because of **latency issues**
 - A group of events might occur **around the same time frame** (similar event times), but some **might arrive later than others** (late ingestion times)
 - should be aware of late arriving data and the **impact on downstream systems and uses**
 - To handle late-arriving data, **need to set a cutoff time for when late-arriving data will no longer be processed**

Message and Stream Ingestion Considerations(2)

Ordering and Multiple Delivery, Replay, Message Size

- Ordering and Multiple Delivery
 - Streaming platforms are generally built out of **distributed systems**, which can cause some complications
 - Specifically, messages may be **delivered out of order** and more than once (**at-least-once delivery**)
- Replay
 - allows readers to **request a range of messages from the history**, allowing to rewind your event history to a particular point in time
 - a key capability in many streaming ingestion platforms and is particularly useful when **need to reingest and reprocess data for a specific time range**
 - For example,
 - RabbitMQ typically deletes messages after all subscribers consume them
 - Kafka, Kinesis, and Pub/Sub all support event retention and replay
- Message Size
 - easily **overlooked** issue - must ensure that the streaming framework in question can handle the maximum expected message size
 - Amazon Kinesis supports a maximum message size of 1 MB
 - Kafka defaults to this maximum size but can be configured for a maximum of 20 MB or more

Message and Stream Ingestion Considerations(3)

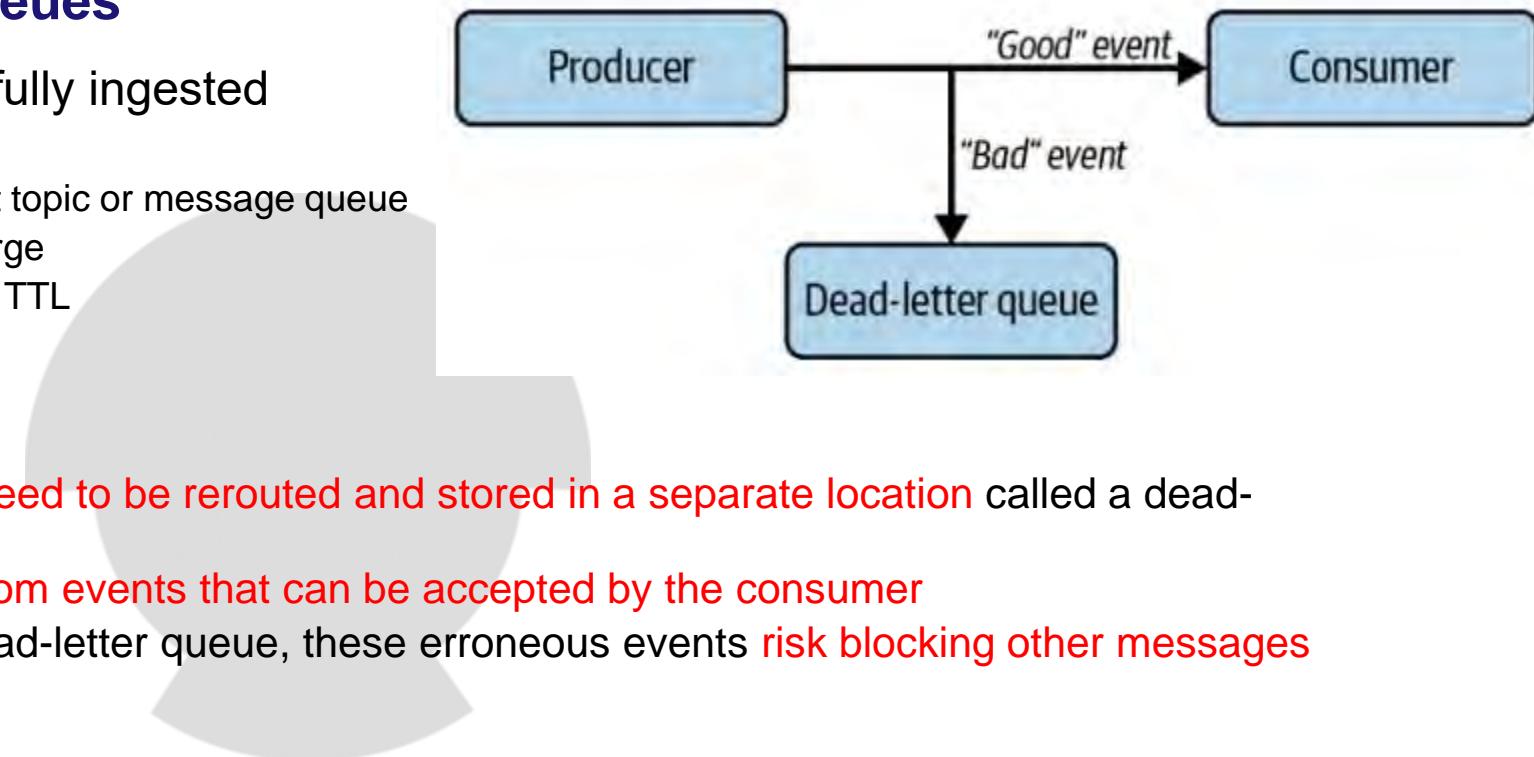
Time to Live

- How long to preserve event record?
- A key parameter is **maximum message retention time**, also known as the time to live (TTL)
 - usually **a configuration set** for how long want events to live before they are acknowledged and ingested
 - Any unacknowledged event that's not ingested after its TTL expires automatically disappears
- Helpful to **reduce backpressure and unnecessary event volume** in event-ingestion pipeline.
 - An **extremely short TTL** (milliseconds or seconds) might cause most messages to **disappear** before processing
 - A **very long TTL** (several weeks or months) will create a **backlog of many unprocessed messages**, resulting in long wait times
- For example,
 - Google Cloud Pub/Sub supports retention periods of up to 7 days
 - Amazon Kinesis Data Streams retention can be turned up to 365 days
 - Kafka can be configured for indefinite retention, limited by available disk space

Message and Stream Ingestion Considerations(4)

Error Handling and Dead-Letter Queues

- Sometimes events aren't successfully ingested
 - Perhaps
 - an event is sent to a nonexistent topic or message queue
 - the message size may be too large
 - or the event has expired past its TTL
- Dead-letter queue
 - Events that cannot be ingested **need to be rerouted and stored in a separate location** called a dead-letter queue
 - **segregates problematic events from events that can be accepted by the consumer**
 - If events are not rerouted to a dead-letter queue, these erroneous events **risk blocking other messages from being ingested**
- Can be used to
 - **diagnose** why event ingestions errors occur and solve data pipeline **problems**
 - might be able to **reprocess some messages** in the queue after fixing the underlying cause of errors



Message and Stream Ingestion Considerations(5)

Consumer Pull and Push

- A consumer subscribing to a topic can get events in two ways: **push and pull**
- Kafka and Kinesis support only **pull subscriptions**
 - Subscribers read messages from a topic and confirm when they have been processed
- In addition to **pull subscriptions**, Pub/Sub and RabbitMQ support **push subscriptions**
 - allowing these services to write messages to a listener
- Pull subscriptions are the default choice for most data engineering applications,
 - but may want to consider push capabilities for specialized applications
 - Pull-only message ingestion systems can still push if an extra layer is added to handle this



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Ways to Ingest Data

Pravin Y Pawar

Adapted from
Fundamentals of Data Engineering
By Joe Reis and Matt Housley

Direct Database Connection

ODBC/ JDBC drivers

- Data can be pulled from databases for ingestion by querying and reading over a network connection
 - most commonly, this connection is made using ODBC or JDBC
- ODBC uses a driver hosted by a client accessing the database to translate commands issued to the standard ODBC API into commands issued to the database
 - Database returns query results over the wire, where the driver receives them and translates them back into a standard form to be read by the client
 - For ingestion, the application utilizing the ODBC driver is an ingestion tool
 - The ingestion tool may pull data through many small queries or a single large query
- JDBC - conceptually remarkably similar to ODBC
 - A Java driver connects to a remote database and serves as a translation layer between the standard JDBC API and the native network interface of the target database
 - The Java Virtual Machine (JVM) is standard, portable across hardware architectures and operating systems
 - provides the performance of compiled code through a just-in-time (JIT) compiler
 - JDBC provides extraordinary database driver portability
- ODBC drivers are shipped as OS and architecture native binaries
 - database vendors must maintain versions for each architecture/OS version that they wish to support
- Vendors can ship a single JDBC driver that is compatible with any JVM language (e.g., Java, Scala, Clojure, or Kotlin) and JVM data framework (i.e., Spark.)

Change data capture (CDC)

Batch vs Continuous CDC

- Process of ingesting changes from a source database system
 - A source PostgreSQL system that supports an application and periodically or continuously ingests table changes for analytics
- Batch-oriented CDC
 - If the database table in question has an `updated_at` field containing the last time a record was written or updated
 - can query the table to find all updated rows since a specified time
 - allows to pull changes and differentially update a target table
 - a key limitation: don't necessarily obtain all changes that were applied to these rows
 - can be mitigated by utilizing an insert-only schema, where each account transaction is recorded as a new record in the table (see "Insert-Only")
- Continuous CDC
 - captures all table history and can support near real-time data ingestion
 - either for real-time database replication or to feed real-time streaming analytics
 - treats each write to the database as an event

Change data capture (CDC) 2

Ways

- Log-based CDC
 - One of the most common approaches with a transactional database such as PostgreSQL D
 - Database binary log records every change to the database sequentially
 - A CDC tool can read this log and send the events to a target, such as the Apache Kafka Debezium streaming platform
- Managed CDC paradigm
 - Some databases support a simplified, managed CDC paradigm
 - Many cloud-hosted databases can be configured to directly trigger a serverless function or write to an event stream every time a change happens in the database
- CDC considerations
 - CDC consumes various database resources, such as memory, disk bandwidth, storage, CPU time, and network bandwidth
 - For batch CDC, running any large batch query against a transactional production system can cause excessive load
 - Either run such queries only at off-hours or use a read replica to avoid burdening the primary database

APIs

Trends

- APIs are a data source that continues to grow in importance and popularity
 - A typical organization may have hundreds of external data sources such as SaaS platforms or partner companies
 - hard reality is that no proper standard exists for data exchange over APIs
 - need to spend a significant amount of time
 - reading documentation
 - communicating with external data owners
 - writing and maintaining API connection code
- Three trends:
 - Many vendors provide API client libraries for various programming languages that remove much of the complexity of API access
 - Numerous data connector platforms are available now as SaaS, open source, or managed open source
 - provide turnkey data connectivity to many data sources
 - offer frameworks for writing custom connectors for unsupported data sources
 - Emergence of data sharing - the ability to exchange data through a standard platform
 - such as BigQuery, Snowflake, Redshift, or S3
 - Once data lands on one of these platforms, it is straightforward to store it, process it, or move it somewhere else

APIs

Considerations

- Data sharing has had a large and rapid impact in the data engineering space
- A managed service might look like an expensive option, consider the value of time and the opportunity cost of building API connectors
 - support building custom API connectors
 - provide API technical specifications in a standard format or writing connector code that runs in a serverless function framework
- Reserve custom connection work for APIs that aren't well supported by existing frameworks
 - will find that there are still plenty of these to work on
 - Two main aspects: software development and ops
 - Follow software development best practices - version control, continuous delivery, and automated testing
 - consider an orchestration framework, which can dramatically streamline the operational burden of data ingestion

Message Queues and Event-Streaming Platforms

- Message queues and event-streaming platforms are widespread ways to ingest real-time data
 - from web and mobile applications, IoT sensors, and smart devices
- Popular real-time data ingestion includes message queues or event-streaming platforms
 - Though these are both source systems, they also act as ways to ingest data
 - In both cases, consume events from the publisher subscribed to
- A message is handled at the individual event level and is meant to be transient
 - Once a message is consumed, it is acknowledged and removed from the queue
- A stream ingests events into an ordered log
 - The log persists for as long as you wish, allowing events to be queried over various ranges, aggregated, and combined with other streams to create new transformations published to downstream consumers
- Batch usually involves static workflows (ingest data, store it, transform it, and serve it)
- Messages and streams are fluid
 - Ingestion can be nonlinear, with data being published, consumed, republished, and reconsumed

Moving Data with Object Storage

- Object storage is a multitenant system in public clouds
 - is the most optimal and secure way to handle file exchange
 - supports storing massive amounts of data
 - makes ideal for moving data in and out of data lakes, between teams, and transferring data between organizations
- Can even provide short-term access to an object with a signed URL, giving a user temporary permission
- Public cloud storage
 - implements the latest security standards
 - has a robust track record of scalability and reliability
 - accepts files of arbitrary types and sizes
 - and provides high-performance data movement

Databases and File Export

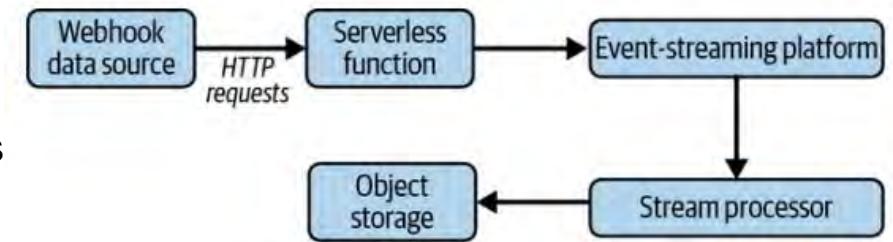
- Source database systems provides file export
 - Export involves large data scans that significantly load the database for many transactional systems
- Source system engineers must
 - assess when these scans can be run without affecting application performance
 - might opt for a strategy to mitigate the load
 - break export queries into smaller exports by querying over key ranges or one partition at a time
 - use a read replica to reduce load
- Major cloud data warehouses are highly optimized for direct file export
 - Snowflake, BigQuery, Redshift, and others support direct export to object storage in various formats

Shell, SSH, SFTP and SCP

- Shell
 - is an interface by which you may execute commands to ingest data
 - can be used to script workflows for virtually any software tool
 - Shell script might
 - read data from a database
 - reserialize it into a different file format
 - upload it to object storage
 - and trigger an ingestion process in a target database
- SSH
 - not an ingestion strategy but a protocol used with other ingestion strategies
 - can be used for file transfer with SCP
 - SSH tunnels are used to allow secure, isolated connections to databases
 - To connect to the database, a remote machine first opens an SSH tunnel connection to the bastion host and then connects from the host machine to the database
- SFTP
 - Accessing and sending data both from secure FTP (SFTP) and secure copy (SCP) are familiar techniques
 - SFTP is still a practical reality for many businesses - work with partner businesses that consume or provide data using SFTP
- SCP
 - To avoid data leaks, security analysis is critical in these situations
 - SCP is a file-exchange protocol that runs over an SSH connection
 - SCP can be a secure file-transfer option if it is configured correctly

Webhooks

- Are often referred to as reverse APIs!
- For a typical REST data API, the data provider gives engineers API specifications that they use to write their data ingestion code
 - The code makes requests and receives data in responses
- With a webhook, the data provider defines an API request specification, but the data provider makes API calls rather than receiving them
 - data consumer's responsibility to provide an API endpoint for the provider to call
 - The consumer is responsible for ingesting each request and handling data aggregation, storage, and processing
- Using appropriate off-the-shelf tools, data engineers can build more robust webhook architectures
 - with lower maintenance and infrastructure costs
- For example, a webhook pattern in AWS might use
 - a serverless function framework (Lambda) to receive incoming events
 - a managed event-streaming platform to store and buffer messages (Kinesis)
 - a stream-processing framework to handle real-time analytics (Flink)
 - an object store for long-term storage (S3)



Other ways

- Web Interface
 - must manually access a web interface, generate a report, and download a file to a local machine
- Web Scraping
 - automatically extracts data from web pages, often by combing the web page's various HTML elements
 - also a murky area where ethical and legal lines are blurry
- Transfer Appliances for Data Migration
 - For massive quantities of data (100 TB or more), transferring data directly over the internet may be a slow and costly process
 - the fastest, most efficient way to move data is not over the wire but by truck
 - Cloud vendors offer the ability to send data via a physical “box of hard drives.”
 - Simply order a storage device, called a transfer appliance, load your data from your servers, and then send it back to the cloud vendor, which will upload data
 - are one-time data ingestion events and are not suggested for ongoing workloads
- Data Sharing
 - is growing as a popular option for consuming data
 - Data providers will offer datasets to third-party subscribers, either for free or at a cost
 - datasets are often shared in a read-only fashion, meaning can integrate these datasets with own data (and other third-party datasets),
 - but you do not own the shared dataset
 - If the data provider decides to remove your access to a dataset, no longer have access to it
 - Many cloud platforms offer data sharing



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Questions About the Data

Pravin Y Pawar

Largely adapted from Machine Learning Engineering
By Andriy Burkov

Need

- The data available to the analyst is
 - not always “right”
 - not always in a **form** that a machine learning **algorithm** can use
- Discuss
 - the properties of good quality data,
 - typical problems a dataset can have
- Questions about data
 - Is the Data Accessible?
 - Is the Data Sizeable?
 - Is the Data Usable?
 - Is the Data Understandable?
 - Is the Data Reliable?

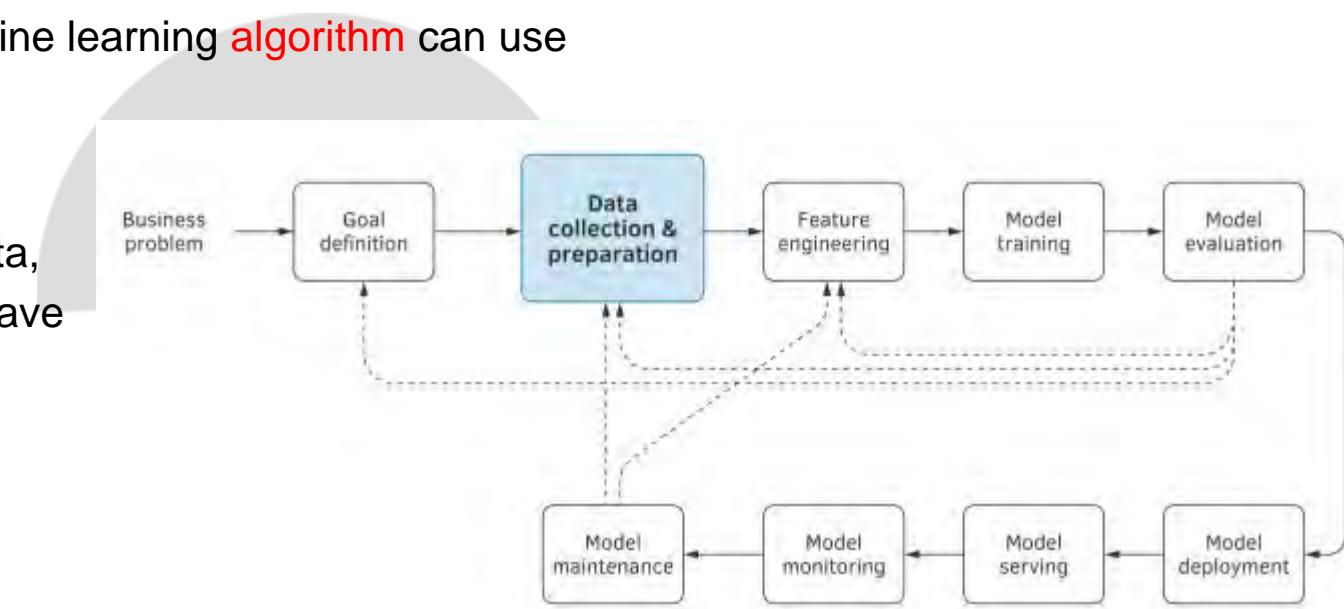


Figure 1: Machine learning project life cycle.

Is the Data Accessible?

- Does the data you need already exist?
 - If yes, is it accessible (physically, contractually, ethically, or from a cost perspective)?
 - If you are purchasing or re-using someone else's data sources, have you considered how that data might be used or shared?
 - Do you need to negotiate a new licensing agreement with the original supplier?
- If the data is accessible, is it protected by copyright or other legal norms?
 - If so, have you established who owns the copyright in your data?
 - Might there be joint copyright?
 - Is the data sensitive (e.g., concerning your organization's projects, clients, or partners, or it is classified by the government), and are there any potential privacy issues? If so, have you
 - discussed data sharing with the respondents from whom you collected the data?
 - Can you preserve personal information for a long-term so that it can be used in the future?
 - Do you need to share the data along with the model? If so, do you need to get written consent from owners or respondents?
 - Do you need to anonymize data?
- Even if it's physically possible to get the data you need, don't work with it until all the above questions are resolved.

Is the Data Sizeable?

- The question for which would like to have a definitive answer is whether there's enough data
 - However, as already found out, it's usually not known how much data is needed to reach the goal,
 - especially if the minimum model quality requirement is stringent
- If have doubts about the immediate availability of sufficient data,
 - need find out how frequently new data gets generated
 - For some projects, can start with what's initially available and, while working on it, new data might gradually come in
- One practical way to find out if have collected sufficient data is to plot learning curves
 - More specifically, plot the training and validation scores of learning algorithm for varying numbers of training examples

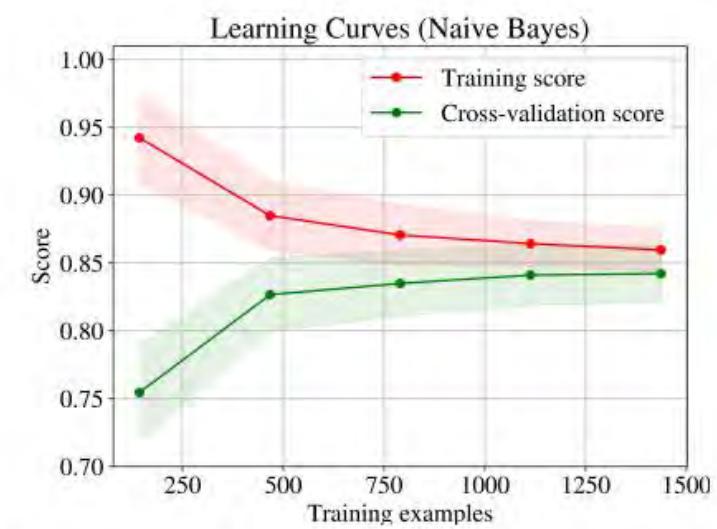


Figure 2: Learning curves for the Naïve Bayes learning algorithm applied to the standard "digits" dataset of scikit-learn.

Is the Data Useable?

- The data quality is one of the major factors affecting the model performance
- Imagine that want to train a model that predicts a person's gender, given their name
 - might acquire a dataset of people that contains gender information
 - however, if use this dataset blindly, might realize that no matter how hard one tries to improve the quality of model
 - its performance on new data is low.
- A tidy dataset can
 - have missing values - consider data imputation techniques to fill the missing values
 - have magic number - 9999 or -1
 - contains duplicates - check why those are added
 - have data which expired or be significantly not up to date
 - have data which is incomplete or unrepresentative of the phenomenon

Is the Data Understandable?

- Crucial to understand from where each attribute in the dataset came
 - equally important to understand what each attribute exactly represents
- One frequent problem observed in practice is when the variable that the analyst tries to predict is found among the features in the feature vector
 - Imagine that working on the problem of predicting the price of a house from its attributes
 - such as the number of bedrooms, surface, location, year of construction, and so on
 - Without spending too much time analyzing each column, remove only the transaction price from the attributes and use that value as the target to be learnt
 - predicts the transaction price with accuracy near 100%
 - Model is delivered to the client, they deploy it in production
 - the tests show that the model is wrong most of the time.
 - What did happen is called data leakage (also known as target leakage)
 - After a more careful examination of the dataset, realize that one of the columns in the spreadsheet contained the real estate agent's commission
 - Of course, the model easily learned to convert this attribute into the house price perfectly
 - However, this information is not available in the production environment before the house is sold

Is the Data Reliable?

- The reliability of a dataset varies depending on the procedure used to gather that dataset
- Can you trust the labels?
 - If the data was produced by the workers on Mechanical Turk , then the reliability of such data might be very low
 - In some cases, the labels assigned to feature vectors might be obtained as a majority vote
 - better to do additional validation of quality on a small random sample of the dataset
- The reliability of labels can also be affected by the delayed or indirect nature of the label.

Is the Data Reliable?(2)

Delayed labels

- The label is considered delayed when the feature vector to which the label was assigned represents something that happened significantly earlier than the time of label observation
- Take the churn prediction problem
 - a feature vector describing a customer, and want to predict whether the customer will leave at some point in the future
 - typically six months to one year from now
 - feature vector represents what we know about the customer now, but the label (“left” or “stayed”) will be assigned in the future
 - This is an important property, because between now and the future, many events, not reflected in our feature vector might happen which would affect the customer’s decision to stay or leave.
 - Therefore delayed labels make our data less reliable

Is the Data Reliable?(3)

Direct or indirect labels

- Whether a label is direct or indirect also affects reliability, depending, of course, on what we are trying to predict.
- For example,
 - let's say goal is to predict whether the website visitor will be interested in a webpage
 - might acquire a certain dataset containing information about users, webpages, and labels “interested”/“not_interested” reflecting whether a specific user was interested in a particular webpage
 - A direct label would indeed indicate interest, while an indirect label could suggest some interest.
 - if the user pressed the “Like” button, have the direct indicator of interest
 - if the user only clicked on the link, this could be an indicator of some interest, but it's an indirect indicator
 - The user could have clicked by mistake, or because the link text was a clickbait, we cannot know for sure
 - If the label is indirect, this also makes such data less reliable
 - Of course, it's less reliable for predicting the interest, but can be perfectly reliable for predicting clicks

Is the Data Reliable?(4)

Feedback loops

- A feedback loop is a property in the system design when the data used to train the model is obtained using the model itself
- Imagine that working on a problem of predicting whether a specific user of a website will like the content
 - only have indirect labels – clicks
- If the model is already deployed on the website and the users click on links recommended by the model
 - means that the new data indirectly reflects not only the interest of users to the content, but also how intensively the model recommended that content
- If the model decided that a specific link is important enough to recommend to many users,
 - more users would likely click on that link, especially if the recommendation was made repeatedly during several days or weeks



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Common Problems With Data

Pravin Y Pawar

Largely adapted from Machine Learning Engineering
By Andriy Burkov

High Cost

Labeling

- Getting unlabeled data can be expensive
 - however, labeling data is the most expensive work, especially if the work is done manually
 - also when it must be gathered specifically for your problem
 - must be assigned manually, and paying someone to do that work is expensive
- Google has a clever technique outsourcing the labeling to random people with its free reCAPTCHA service
 - solves two problems:
 - reducing spam on the Web
 - providing cheap labeled data to Google
- The goal of a good labeling process design is to make the labeling as streamlined as poss



Figure 3: The unlabeled and labeled aerial photo. Photo credit: Tom Fisk.

High Cost(2)

Tool-based labeling

- Well-designed labeling tools
 - will minimize mouse use (including menus activated by mouse clicks), maximize hotkeys,
 - reduce costs by increasing the speed of data labeling.
- Whenever possible, reduce decision-making to a yes/no answer
 - If the labeler clicks “Not Sure,” can save this example to analyze later or simply not use such examples for training the model
- Another trick allowing for accelerated labeling is noisy pre-labeling consisting of prelabeling the example using the current best model
 - start by labeling a certain quantity of examples “from scratch” (that is, without using any support)
 - build the first model that works reasonably well, using this initial set of labeled examples
 - Next, use the current model and label each new example in place of the human labeler
 - Ask whether the automatically assigned label is correct
 - If the labeler clicks “Yes,” save this example as usual
 - If they click “No,” then ask to label this example manually

Bad Quality

- Data quality is one of the major factors affecting the performance of the model
- Data quality has two components:
 - raw data quality and labeling quality
- Some common problems with raw data are
 - noise,
 - bias,
 - low predictive power,
 - outdated examples,
 - outliers,
 - leakage

Noise

- Noise in data is a corruption of examples
 - Images can be blurry or incomplete
 - Text can lose formatting, which makes some words concatenated or split
 - Audio data can have noise in the background
 - Poll answers can be incomplete or have missing attributes, such as the responder's age or gender
- Noise is often a random process that corrupts each example independently of other examples in the collection
 - If tidy data has missing attributes, data imputation techniques can help in guessing values for those attributes
 - Blurred images can be deblurred using specific image deblurring algorithms
 - Noise in audio data can be algorithmically suppressed
- Noise is more a problem when the dataset is relatively small (thousands of examples or less),
 - because the presence of noise can lead to overfitting
 - the algorithm may learn to model the noise contained in the training data, which is undesirable
- In the big data context, on the other hand, noise is typically “averaged out” over multiple examples

Bias

- Bias in data is an inconsistency with the phenomenon that data represents
 - occur for a number of reasons (which are not mutually exclusive)
- Types of Bias
 - Selection bias
 - Omitted variable bias
 - Sponsorship or funding bias
 - Sampling bias (also known as distribution shift)
 - Prejudice or stereotype bias
 - Systematic value distortion
 - Experimenter bias
 - Labeling bias

Low Predictive Power

- Is an issue that's often not considered until one has spent fruitless energy trying to train a good model
- Don't know whether
 - the model underperform because it is not expressive enough?
 - the data not contain enough information from which to learn?
- Suppose the goal is to predict whether a listener will like a new song on a music streaming service
 - data is the name of the artist, the song title, lyrics, and whether that song is in their playlist
 - model trained with this data will be far from perfect!
- Artists who are not in the listener's playlist are unlikely to receive a high score from the model
 - many users will only add some songs of a specific artist to their playlist
 - musical preferences are significantly influenced by the song arrangement, choice of instruments, sound effects, tone of voice, and subtle changes in tonality, rhythm, and beat
 - These are properties of songs that cannot be found in lyrics, title, or the artist's name; they have to be extracted from the sound file

Outdated Examples

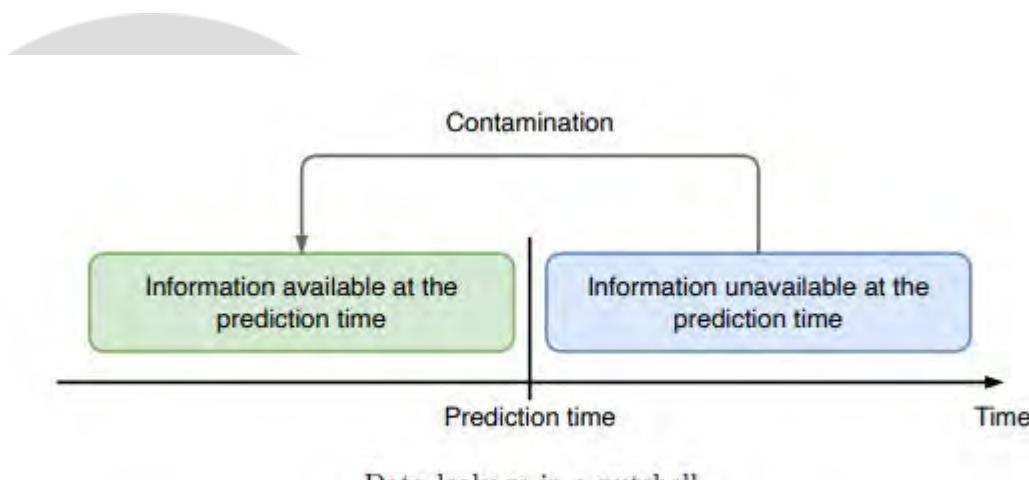
- Once you build the model and deploy it in production, the model usually performs well for some time!
 - a certain model quality monitoring procedure is deployed in the production environment
 - Once an erratic behavior is detected, new training data is added to adjust the model; the model is then retrained and redeployed
- Often, the cause of an error is explained by the finiteness of the training set.
 - In such cases, additional training examples will solidify the model
 - However, in many practical scenarios, the model starts to make errors because of concept drift.
- Concept drift is a fundamental change in the statistical relationship between features and label
- Imagine your model predicts whether a user will like certain content on a website
 - Over time, the preferences of some users may start to change, perhaps due to aging, or because a user discovers something new
 - examples added to the training data in the past no longer reflect some user's preferences and start hurting the model performance
 - rather than contributing to it - aka concept drift

Outliers

- Outliers are examples that look dissimilar to the majority of examples from the dataset
 - up to the data analyst to define “dissimilar”
 - Typically, measured by some distance metric, such as Euclidean distance
- Shallow algorithms are particularly sensitive to outliers
 - linear or logistic regression, and some ensemble methods, such as AdaBoost
- Debatable
 - Whether to exclude outliers from the training data, or to use machine learning algorithms and models robust to outliers
 - Deleting examples from a dataset is not considered scientifically or methodologically sound, especially in small datasets
 - In the big data context, outliers don't typically have a significant influence on the model
- From a practical standpoint, if excluding some training examples results in better performance of the model on the holdout data,
 - the exclusion may be justified
- Which examples to consider for exclusion can be decided based on a certain similarity measure

Data Leakage

- Data leakage, also called target leakage,
 - a problem affecting several stages of the machine learning life cycle, from data collection to model evaluation
- Data leakage in supervised learning is the unintentional introduction of information about the target that should not be made available
 - also called as “contamination”
 - Training on contaminated data leads to overly optimistic expectations about the model performance



Data leakage in a nutshell.



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

What Is Good Data?

Pravin Y Pawar

Largely adapted from Machine Learning Engineering
By Andriy Burkov

The properties of good data

- Good data
 - contains **enough information** that can be used for modeling,
 - has **good coverage** of what you want to do with the model,
 - reflects **real inputs** that the model will see in production,
 - is as **unbiased** as possible,
 - is **not a result of the model itself**,
 - has **consistent labels**, and
 - is **big enough** to allow **generalization**

Good Data Is Informative

- Contains enough information that can be used for modeling
- For example,
 - Lets say you want to train a model that predicts whether the customer will buy a specific product,
 - then you need to possess both the properties of the product in question and the properties of the products customers purchased in the past
 - If only have the properties of the product and a customer's location and name, then the predictions will be the same for all users from the same location.
- If have enough training examples, then
 - the model can potentially derive the gender and ethnicity from the name
 - make different predictions for men, women, locations, and ethnicities, but not to each customer individually

Good Data Has Good Coverage

- Good data has good coverage of what you want to do with the model.
- For example,
- If going to use the model to classify web pages by topic and have a thousand topics of interest,
 - then data has to contain examples of documents on each of the thousand topics in quantity sufficient for the algorithm to be able to learn the difference between topics
- Imagine a different situation.
 - Let's say that for a particular topic, only have one or a couple of documents
 - Let each document contain a unique ID in the text
 - In such a scenario, the learning algorithm will not be sure what it must look at in each document to understand to which topic it belongs. Maybe the IDs? They look like good differentiators!
 - If the algorithm decides to use IDs to separate these couple examples from the rest of the dataset, then the learned model will not be able to generalize: it will not see any of those IDs ever again.

Good Data Reflects Real Inputs

- Good data reflects real inputs that the model will see in production
- For example,
 - If building a system that recognizes cars on the road and all pictures have been taken during the working hours,
 - then it's unlikely that will have many examples of night pictures
 - Once the model is deployed in production, pictures will start coming from all times of the day,
 - and model will more frequently make errors on night pictures

Good Data Is Unbiased

- Good data is as unbiased as possible - looks similar to the previous one
 - Still, bias can be present in both the data used for training and the data that the model is applied to in the production environment
- A user interface can also be a source of bias.
- For example,
 - Want to predict the popularity of a news article, and use the click rate as a feature
 - If some news article was displayed on the top of the page, the number of clicks it got would often be higher compared to another news article displayed on the bottom, even if the latter is more engaging

Good Data Is Not a Result of a Feedback Loop

- Good data is not a result of the model itself - echoes the problem of the feedback loop!
- For example,
 - Cannot train a model that predicts the gender of a person from their name, and then use the prediction to label a new training example
 - Alternatively, if model is used to decide which email messages are important to the user and highlight those important messages,
 - one should not directly take the clicks on those emails as a signal that the email is important
 - The user might have clicked on them because the model highlighted them.

Good Data Has Consistent Labels

- Good data has consistent labels
- Inconsistency in labeling can come from several sources:
 - Different people do labeling according to different criteria
 - Even if people believe that they use the same criteria, different people often interpret the same criteria differently.
 - The definition of some classes evolved over time
 - results in a situation when two very similar feature vectors receive two different labels
 - Misinterpretation of user's motives
 - user ignored a recommended news article, as a consequence, this news article receives a negative label
 - However, the motive of the user for ignoring this recommendation might be that they already knew the story and not that they are uninterested in the topic of the story.

Good Data Is Big Enough

- Good data is big enough to allow generalization
- Sometimes, nothing can be done to increase the accuracy of the model
 - No matter how much data you throw on the learning algorithm: the information contained in the data has low predictive power for problem.
- However, more often, can get a very accurate model if pass from thousands of examples to millions or hundreds of millions
 - cannot know how much data is needed before started working on the problem and see the progress



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Bias and Fairness

Pravin Y Pawar

Adapted from Google Machine Learning Course

Fairness

- Evaluating a machine learning model responsibly requires doing more than just calculating loss metrics
 - critical to audit training data and evaluate predictions for bias
- Machine learning models are not inherently objective
 - Engineers train models by feeding them a data set of training examples
 - This human involvement in the provision and curation of this data can make a model's predictions susceptible to bias.
- Important to be aware of common human biases that can manifest in data
 - proactive steps can be taken to mitigate their effects

Types of Bias

Reporting Bias

- Occurs when the frequency of events, properties, and/or outcomes captured in a data set does not accurately reflect their real-world frequency
 - Can arise because people tend to focus on documenting circumstances that are unusual or especially memorable
-
- EXAMPLE:
 - A sentiment-analysis model is trained to predict whether book reviews are positive or negative based on a corpus of user submissions to a popular website. The majority of reviews in the training data set reflect extreme opinions (reviewers who either loved or hated a book), because people were less likely to submit a review of a book if they did not respond to it strongly. As a result, the model is less able to correctly predict sentiment of reviews that use more subtle language to describe a book.

Types of Bias

Automation Bias

- Tendency to favor results generated by automated systems over those generated by non-automated systems, irrespective of the error rates of each
- EXAMPLE:
- Software engineers working for a sprocket manufacturer were eager to deploy the new "groundbreaking" model they trained to identify tooth defects, until the factory supervisor pointed out that the model's precision and recall rates were both 15% lower than those of human inspectors.

Types of Bias

Selection Bias

- Selection bias occurs if a data set's examples are chosen in a way that is not reflective of their real-world distribution
- Coverage bias: Data is not selected in a representative fashion.
 - EXAMPLE: A model is trained to predict future sales of a new product based on phone surveys conducted with a sample of consumers who bought the product. Consumers who instead opted to buy a competing product were not surveyed, and as a result, this group of people was not represented in the training data.
- Non-response bias (or participation bias): Data ends up being unrepresentative due to participation gaps in the data-collection process.
 - EXAMPLE: A model is trained to predict future sales of a new product based on phone surveys conducted with a sample of consumers who bought the product and with a sample of consumers who bought a competing product. Consumers who bought the competing product were 80% more likely to refuse to complete the survey, and their data was underrepresented in the sample.
- Sampling bias: Proper randomization is not used during data collection.
 - EXAMPLE: A model is trained to predict future sales of a new product based on phone surveys conducted with a sample of consumers who bought the product and with a sample of consumers who bought a competing product. Instead of randomly targeting consumers, the surveyer chose the first 200 consumers that responded to an email, who might have been more enthusiastic about the product than average purchasers.

Types of Bias

Group Attribution Bias

- Tendency to generalize what is true of individuals to an entire group to which they belong
- Two key manifestations are:
- In-group bias
 - A preference for members of a group to which you also belong, or for characteristics that you also share.
 - EXAMPLE: Two engineers training a résumé-screening model for software developers are predisposed to believe that applicants who attended the same computer-science academy as they both did are more qualified for the role.
- Out-group homogeneity bias
 - A tendency to stereotype individual members of a group to which you do not belong, or to see their characteristics as more uniform.
 - EXAMPLE: Two engineers training a résumé-screening model for software developers are predisposed to believe that all applicants who did not attend a computer-science academy do not have sufficient expertise for the role.

Types of Bias

Implicit Bias

- Occurs when assumptions are made based on one's own mental models and personal experiences that do not necessarily apply more generally
- EXAMPLE:
 - An engineer training a gesture-recognition model uses a head shake as a feature to indicate a person is communicating the word "no." However, in some regions of the world, a head shake actually signifies "yes."
- Confirmation bias
 - Model builders unconsciously process data in ways that affirm preexisting beliefs and hypotheses
- Experimenter's bias
 - In some cases, a model builder may actually keep training a model until it produces a result that aligns with their original hypothesis
- EXAMPLE:
 - An engineer is building a model that predicts aggressiveness in dogs based on a variety of features (height, weight, breed, environment). The engineer had an unpleasant encounter with a hyperactive toy poodle as a child, and ever since has associated the breed with aggression. When the trained model predicted most toy poodles to be relatively docile, the engineer retrained the model several more times until it produced a result showing smaller poodles to be more violent.

Fairness: Identifying Bias

Missing Feature Values

- If your data set has one or more features that have missing values for a large number of examples, that could be an indicator that certain key characteristics of your data set are under-represented.
- For example, the table below shows a summary of key stats for a subset of features in the California Housing dataset
- Suppose instead that three features (population, households, and median_income) only had a count of 3000—in other words, that there were 14,000 missing values for each feature.
- These 14,000 missing values would make it much more difficult to accurately correlate average income of households with median house prices
- Before training a model on this data, it would be prudent to investigate the cause of these missing values to ensure that there are no latent biases responsible for missing income and population data.

	longitude	latitude	total_rooms	population	households	median_income	median_house_value
count	17000.0	17000.0	17000.0	3000.0	3000.0	3000.0	17000.0
mean	-119.6	35.6	2643.7	1429.6	501.2	3.9	207.3
std	2.0	2.1	2179.9	1147.9	384.5	1.9	116.0
min	-124.3	32.5	2.0	3.0	1.0	0.5	15.0
25%	-121.8	33.9	1462.0	790.0	282.0	2.6	119.4
50%	-118.5	34.2	2127.0	1167.0	409.0	3.5	180.4
75%	-118.0	37.7	3151.2	1721.0	605.2	4.8	265.0
max	-114.3	42.0	37937.0	35682.0	6082.0	15.0	500.0

Fairness: Identifying Bias

Unexpected Feature Values

- When exploring data, you should also look for examples that contain feature values that stand out as especially uncharacteristic or unusual
- These unexpected feature values could indicate problems that occurred during data collection or other inaccuracies that could introduce bias.
- Can you pinpoint any unexpected feature values?

	longitude	latitude	total_rooms	population	households	median_income	median_house_value
1	-121.7	38.0	7105.0	3523.0	1088.0	5.0	0.2
2	-122.4	37.8	2479.0	1816.0	496.0	3.1	0.3
3	-122.0	37.0	2813.0	1337.0	477.0	3.7	0.3
4	-103.5	43.8	2212.0	803.0	144.0	5.3	0.2
5	-117.1	32.8	2963.0	1162.0	556.0	3.6	0.2
6	-118.0	33.7	3396.0	1542.0	472.0	7.4	0.4

Fairness: Identifying Bias

Data Skew

- Any sort of skew in your data, where certain groups or characteristics may be under- or over-represented relative to their real-world prevalence, can introduce bias into your model.
- Figure visualizes a subset of data drawn from the full housing data set that exclusively represents the northwest region of California
- If this unrepresentative sample were used to train a model to predict California housing prices statewide, the lack of housing data from southern portions of California would be problematic
- The geographical bias encoded in the model might adversely affect homebuyers in unrepresented communities



Fairness: Evaluating for Bias

Example

When evaluating a model, metrics calculated against an entire test or validation set don't always give an accurate picture of how fair the model is.

Consider a new model developed to predict the presence of tumors that is evaluated against a validation set of 1,000 patients' medical records. 500 records are from female patients, and 500 records are from male patients. The following [confusion matrix](#) summarizes the results for all 1,000 examples:

True Positives (TPs): 16	False Positives (FPs): 4
False Negatives (FNs): 6	True Negatives (TNs): 974

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{16}{16 + 4} = 0.800$$

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{16}{16 + 6} = 0.727$$

These results look promising: precision of 80% and recall of 72.7%.

Fairness: Evaluating for Bias(2)

Example Continued

But what happens if we calculate the result separately for each set of patients?

Let's break out the results into two separate confusion matrices: one for female patients and one for male patients.

Female Patient Results

True Positives (TPs): 10	False Positives (FPs): 1
False Negatives (FNs): 1	True Negatives (TNs): 488

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{10}{10 + 1} = 0.909$$

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{10}{10 + 1} = 0.909$$

Male Patient Results

True Positives (TPs): 6	False Positives (FPs): 3
False Negatives (FNs): 5	True Negatives (TNs): 486

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{6}{6 + 3} = 0.667$$

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{6}{6 + 5} = 0.545$$

When we calculate metrics separately for female and male patients, we see stark differences in model performance for each group.

Fairness: Evaluating for Bias(3)

Example Continued

Female patients:

- Of the 11 female patients who actually have tumors, the model correctly predicts positive for 10 patients (recall rate: 90.9%). In other words, **the model misses a tumor diagnosis in 9.1% of female cases**.
- Similarly, when the model returns positive for tumor in female patients, it is correct in 10 out of 11 cases (precision rate: 90.9%); in other words, **the model incorrectly predicts tumor in 9.1% of female cases**.

Male patients:

- However, of the 11 male patients who actually have tumors, the model correctly predicts positive for only 6 patients (recall rate: 54.5%). That means **the model misses a tumor diagnosis in 45.5% of male cases**.
- And when the model returns positive for tumor in male patients, it is correct in only 6 out of 9 cases (precision rate: 66.7%); in other words, **the model incorrectly predicts tumor in 33.3% of male cases**.

We now have a much better understanding of the biases inherent in the model's predictions, as well as the risks to each subgroup if the model were to be released for medical use in the general population.



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

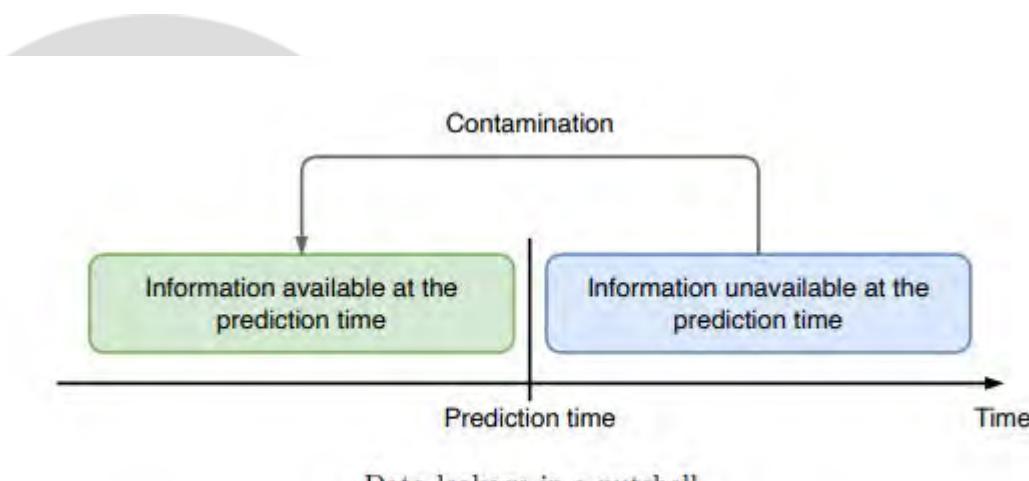
Data Leakage

Pravin Y Pawar

Largely adapted from Machine Learning Engineering
By Andriy Burkov

Data Leakage

- Data leakage, also called target leakage,
 - a problem affecting several stages of the machine learning life cycle, from data collection to model evaluation
- Data leakage in supervised learning is the unintentional introduction of information about the target that should not be made available
 - also called as “contamination”
 - Training on contaminated data leads to overly optimistic expectations about the model performance



Data leakage in a nutshell.

Causes of Data Leakage

- Three most frequent causes of data leakage that can happen during data collection and preparation:
 - target being a function of a feature,
 - feature hiding the target,
 - feature coming from the future

Target being a function of a feature

Example

- Let goal be to predict a country's GDP based on various attributes: area, population, geographic region, and so on
 - Gross Domestic Product (GDP) is defined as the monetary measure of all finished goods and services in a country within a specific period
 - If you don't do a careful analysis of each attribute and its relation to GDP, you might let a leakage happen:
 - two columns, Population and GDP per capita, multiplied, equal GDP
 - The model will perfectly predict GDP by looking at these two columns only
 - The fact that GDP be one of the features, though in a slightly modified form (devised by the population), constitutes contamination and, therefore, leads to data leakage.
- Imagine training a model to predict the yearly salary, given the attributes of an employee
 - The training data is a table that contains both monthly and yearly salary, among many other attributes
 - If monthly salary is not removed from the list of features, that attribute alone will perfectly predict the yearly salary,
 - making believe that model is perfect
 - Once the model is put in production, it will likely stop receiving information about a person's monthly salary:
 - otherwise, the modeling would not be needed

Country	Population	Region	...	GDP per capita	GDP
France	67M	Europe	...	38,800	2.6T
Germany	83M	Europe	...	44,578	3.7T
...
China	1386M	Asia	...	8,802	12.2T

An example of the target (GDP) being a simple function of two features: Population and GDP per capita.

Feature hiding the target

- Sometimes the target is not a function of one or more features, but rather is “hidden” in one of the features.
- Using customer data to predict their gender
 - Look at the column Group
 - If you closely investigate the data in the column Group, represents a demographic value to which each existing customer was related in the past.
 - If the data about a customer’s gender and age is factual, the the column Group constitutes a form of data leakage
 - the value want to predict is “hidden” in the value of a feature
- On the other hand, if the Group values are predictions provided by another, possibly less accurate model
 - then it can be used to build a potentially stronger model - model stacking

Customer ID	Group	Yearly Spendings	Yearly Pageviews	...	Gender
1	M18-25	1350	11,987	...	M
2	F25-35	2365	8,543	...	F
...
18879	F65+	3653	6,775	...	F

An example of the target being hidden in one of the features.

Feature coming from the future

- Feature from the future is a kind of data leakage that is hard to catch
 - if don't have a clear understanding of the business goal
- Imagine a client asked to train a model that predicts whether a borrower will pay back the loan
 - based on attributes such as age, gender, education, salary, marital status, and so on
 - might decide to use all available attributes to predict the value in the column Will Pay Loan,
 - including the data from the column Late Payment Reminders
- Model will look accurate at testing time and model doesn't work well in the production environment
 - in the production environment, the value of Late Payment Reminders is always zero
 - makes sense because the client uses model before the borrower gets the credit, so no reminders have yet been made!
- However, model most likely learned to make the “No” prediction when Late Payment Reminders is 1 or more and pays less attention to the other features
 - Understanding the business context in which the model will be used is, thus, crucial to avoid data leakage.

Borrower ID	Demographic Group	Education	...	Late Payment Reminders	Will Pay Loan
1	M35-50	High school	...	0	Y
2	F25-35	Master's	...	1	N
...
65723	M25-35	Master's	...	3	N

A feature unavailable at the prediction time: Late Payment Reminders.



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Data Skew and Drift

Pravin Y Pawar

Largely adapted from
[Analyzing training-serving skew with TensorFlow](#)
[Data Validation](#)

Production ML is different!

- Machine learning (ML) in production is quite different from ML in a Kaggle competition
 - One of the biggest difference is in the **data sets**
- Kaggle data sets are always **static**
 - given a fixed data set for training and you will be evaluated on a fixed data during testing/evaluation.
- Production data tends to be **constantly changing** over different dimension
 - i.e. time-wise and system-wise
- Monitoring the predictive performance of an ML model in production has emerged as a crucial area of MLOps
- Two common causes of decay in a model's predictive performance over time are the following:
 - Data Drift
 - Concept drift

Data drift

- Data drift
 - A skew grows between training data and serving data
- In data drift, the production data that a model receives for scoring has diverged from the dataset that was used to train, tune, and evaluate the model
- The discrepancies between training data and serving data can usually be classified as
 - Schema skew
 - Distribution skew

Schema skew

- Schema skew occurs when training data and serving data don't conform to the same schema
 - often caused by faults or changes in the upstream process that generates the serving data
- Schema deviations between training and serving data can include the following:
- Inconsistent features
 - a new feature is added to the serving data
- Inconsistent feature types
 - a numerical feature that was an integer number in training data is a real number in serving data
- Inconsistent feature domains
 - a value in a categorical feature disappears, or there's a change in the range of numerical features

Distribution skew

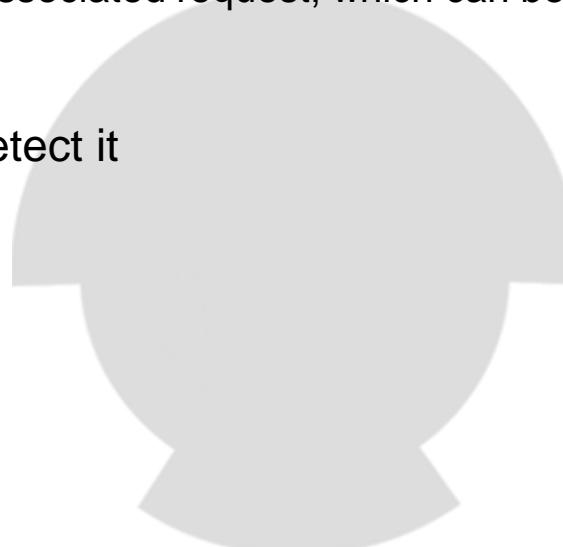
- Distribution skew occurs when the distribution of feature values for training data is significantly different from serving data
 - can be the result of choosing the wrong training dataset to represent real-world data
 - can also happen naturally as new trends and patterns emerge in the data due to the changes in the dynamics of the environment
- Examples include
 - changes in the prices of real estate
 - a change in the popularity of fashion items

Concept drift

- Concept drift means that interpretation of the data has changed
 - As interpretation of the relationship between the input predictors and the target feature evolves
- Often, implies that
 - the mapping of input features to labels that are used during training is no longer valid,
 - or that a novel class or a range of label values has appeared
- Often the result of a change in the process you're attempting to model
 - can also be an evolution of your understanding of this process

Monitoring and detecting drifts and skews

- Prerequisites
 - The system needs to log all incoming features and predictions of the request and response
 - Log the ground truth as well of the associated request, which can be used as additional training data (this is for detecting label drift)
- Use statistical methods to actually detect it
- Supervised monitoring
 - Statistical Process Control
 - Sequential Analysis
 - Adaptive windowing
- Unsupervised monitoring
 - Clustering/novelty detection
 - Feature distribution monitoring
- Model dependent monitoring
 - MD3 – Margin Density Drift Detection



[Source: Production ML: Skews, Data Drift, and Concept Drift](#)

Training-Serving skew

“Past performance is no guarantee of future results.”

- Model decay or drift
 - ML models in production can experience reduced performance over time
 - not only due to being fed bad data and poor programming
 - but also due to datasets and profiles that are constantly evolving
- Natural occurrence in ML models and the speed of decay can vary greatly
 - In some models, it can take years
 - In others, it can happen over the course of a few days
- One of the biggest post-production problems that can lead to an expedited rate of decay is **data-serving skew**
 - a problem that can arise quite easily and be difficult to detect

[What is training-serving skew in Machine Learning?](#)

Training-Serving skew

- Training-serving skew is a difference between ML model outputs during the training and during serving (deployment)
 - essentially a discrepancy between an ML model's feature engineering code during training and during deployment
- Can be caused by:
 - A discrepancy between how data is handled in the training and serving pipelines
 - A change in the data between training and serving
 - A feedback loop between model and algorithm
- Very easy for training-serving skew to crop up
 - via a bug in model's code and cause serious repercussions further down the line, potentially stopping model from working entirely

[What is training-serving skew in Machine Learning?](#)

Avoiding training-serving skew

- Feature Engineering Code
 - Ideally, engineers should be re-using the same feature engineering code
 - to ensure that any given raw data input maps to the same feature vector during training and deployment (serving)
- Computational resources
 - One of the most common reasons for this skew is a mismatch in computational resources at training and deployment time

[What is training-serving skew in Machine Learning?](#)



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Data Validation Approaches

Pravin Y Pawar

Largely Adapted from
["Data Validation in Machine Learning is imperative, not optional"](#)

Data validation

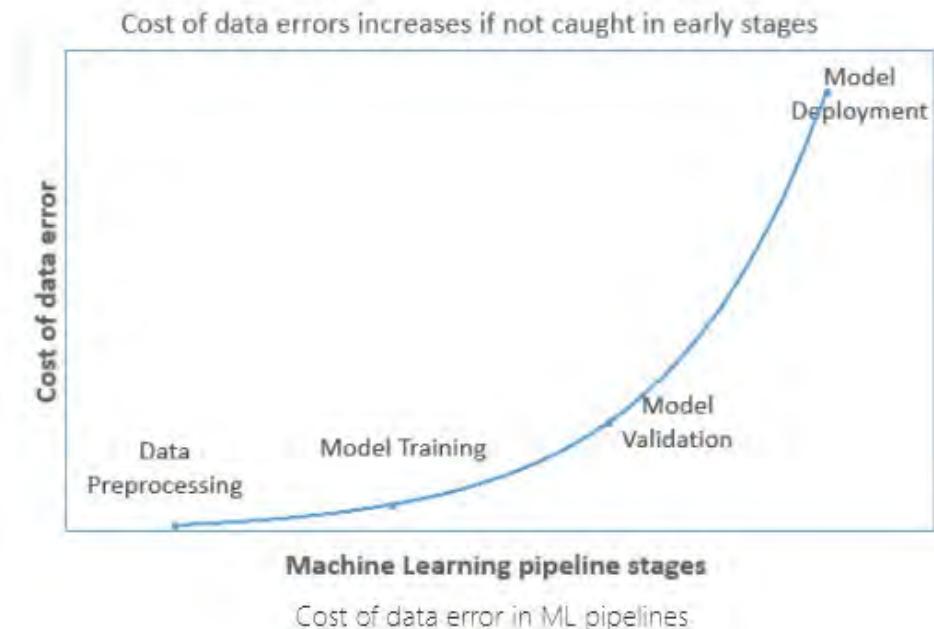
Definition

- Data Validation
 - means checking the accuracy and quality of source data before training a new model version
 - ensures that anomalies that are infrequent or manifested in incremental data are not silently ignored
 - focuses on checking that the statistics of the new data are as expected (e.g. feature distribution, number of categories, etc)
- Examples of such validations in the machine learning pipeline:
 - Are there any **anomalies or data errors** in the incremental data?
 - Are there any **assumptions on data** that are taken **during model training** and are getting violated during serving?
 - Are there **significant differences between training and serving data**?
 - Are there differences between successive data that are getting added into training data?
- Output from the data validation steps should be **informative** enough for a data engineer to take action
 - needs to have **high precision** as too many false alarms will easily be lost credibility

Data validation(2)

Requirement

- “garbage in garbage out”
 - Machine learning models are vulnerable to poor data quality
- In Production
 - model gets re-trained with a fresh set of incremental data added periodically (as frequent as daily)
 - updated model is pushed to the serving layer
 - deployed model makes predictions with new incoming data while serving and the same data is added with actual labels and used for retraining
 - ensures that the newly generated model adapts to the changes in data characteristics
- **New incoming data in the serving layer can change due to various reasons**
 - with time, the erroneous ingested data will become part of the training data, which will start degrading the model accuracy
- **Catching the data errors at an early stage is very important**
 - will reduce the cost of data error which is bound to increase as the error propagates further in the pipeline



Challenges with the data validation

- Creating data validation rules for a dataset with few columns does sound simple
 - when the number of columns in datasets increases, it becomes a humongous task
- Tracking and comparing metrics from the historical datasets to find anomalies in historical trends for each column needs a good amount of recurring time from a data scientist
- Today applications are expected to run 24-by-7 and in such scenarios, data validation needs to be automated
 - data validation component should be smart enough to refresh validation rules

Working of data validation component

- Data validation component serves as a guard post of the ML application
 - does not let bad quality data in
 - keeps a check on each and every new data entry that is going to add to the training data



- Steps
 - Calculate the statistics from the training data against a set of rules
 - Calculate the statistics of the ingested data that has to be validated
 - Compare the statistics of the validation data with the statistics from the training data
 - Store the validation results and takes automated actions like removing the row, capping, or flooring the values
 - Sending the notification and alerts for approval

Unit-test approach by Amazon Research

Deequ

- In software engineering, engineers write unit tests to test their code
 - Similarly, unit tests should also be defined to test the incoming data
- AWS Deequ framework follows the below principles
- Declare constraints:
 - User defines how their data should look like
 - by declaring checks on their data by composing constraints on various columns
- Compute metrics
 - Based on the declared constraint, translate them to measurable metrics
 - metrics can be computed and compared between the data in hand and the incremental data
- Analyze and report
 - Based on the collected metrics over time, predict if the metric on the incremental data is an anomaly or not
 - As a rule, the user can have the system issue “a warning” if the new metric is more than three standard deviations away from the previous mean or throw “an error” if it is more than four standard deviations away.
 - Basis the analysis, report the constraints that fail including the value(s) that made the constraint fail

Unit-test approach by Amazon Research (2)

Deeqe - Constraints

Table 1: Constraints available for composing user-defined data quality checks.

constraint	arguments	semantic
dimension completeness		
isComplete	column	check that there are no missing values in a column
hasCompleteness	column, udf	custom validation of the fraction of missing values in a column
dimension consistency		
isUnique	column	check that there are no duplicates in a column
hasUniqueness	column, udf	custom validation of the unique value ratio in a column
hasDistinctness	column, udf	custom validation of the unique row ratio in a column
isInRange	column, value range	validation of the fraction of values that are in a valid range
statistics (can be used to verify dimension consistency)		
hasSize	udf	custom validation of the number of records
hasTypeConsistency	column, udf	custom validation of the maximum fraction of values of the same data type
hasCountDistinct	column	custom validation of the number of distinct non-null values in a column
hasApproxCountDistinct	column, udf	custom validation of the approx. number of distinct non-null values
hasMin	column, udf	custom validation of a column's minimum value
time		
hasNoAnomalies	metric, detector	validation of anomalies in time series of metric values

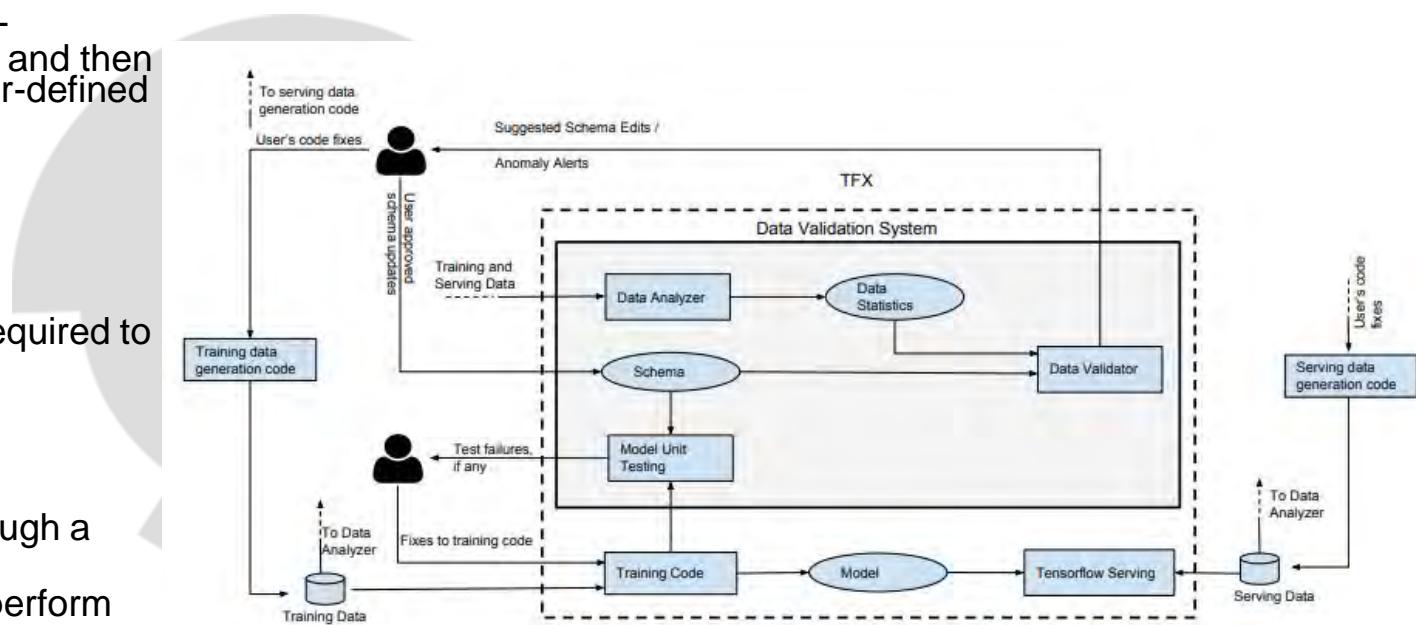
Table 2: Computable metrics to base constraints on.

metric	semantic
dimension completeness	
Completeness	fraction of non-missing values in a column
dimension consistency	
Size	number of records
Compliance	ratio of columns matching predicate
Uniqueness	unique value ratio in a column
Distinctness	unique row ratio in a column
statistics (can be used to verify dimension consistency)	
Minimum	minimal value in a column
Maximum	maximal value in a column
Mean	mean value in a column
StandardDeviation	standard deviation of the value distribution in a column

Data schema approach for data validation by Google Research

Tensorflow data validation

- Google Research has come up with a very similar technique
 - adopted “battle-tested” principles from the data management system and customized it for ML
 - first codifies the expectation from correct data and then using these expected statistics along with user-defined validation schema performs data validation
- Components
- Data Analyzer
 - computes a predefined set of data statistics required to define the data
- Data Validator
 - checks for properties of data as specified through a schema
 - schema is a precursor for a data validator to perform
 - schema list out all the constraints on the features for basic checks and ML-related checks
- Model Unit Tester
 - checks for errors in training code using synthetic data generated through the schema



Differences between Deequ (Amazon) Tensorflow data validation (Google)

Deequ (Amazon)

1 No visualization available

Recalculates training statistics by aggregating
2 prior saved training statistics and new data
statistics.

Provides capability to do anomaly detection
3 based on running average and standard
deviation in addition to the threshold or
absolute/relative difference from training data.

4 Supports data only in SparkDataFrame.

Tensorflow data validation (Google)

Provides visualization using Google
Facets. It summarises statistics for
each feature and compares the trainin
and validation data.

Calculates statistics on whole training
data in every run unless specified. Thi
may become computationally
expensive.

Provides capability to do anomaly
detection based on the threshold or
absolute/relative difference from
training data.

Supports pandas dataframe, CSV, and
best works with TFRecord.

References

[S. Schelter, D. Lange, P. Schmidt, M. Celikel, F. Viessmann and A. Grafberger, “Automating Large-Scale Data Quality Verification” in Proceedings of the VLDB endowment, volume 11, Issue 12: 1781-1794, 2018.](#)

[E. Breck, M. Zinkevich, N. Polyzotis, S. Whang and S. Roy, “Data validation for machine learning”, in Proceedings of the 2nd SysML Conference, Palo Alto, CA, USA, 2019](#)



Thank You!

In our next session:

Automating Large-Scale Data Quality Verification

Sebastian Schelter, Dustin Lange, Philipp Schmidt, Meltem Celikel, Felix Biessmann
Amazon Research
{sseb,langed,phschmid,celikelm,biessmann}@amazon.com

Andreas Grafberger^{*}
University of Augsburg
andreas.grafberger@student.uni-augsburg.de

ABSTRACT

Modern companies and institutions rely on data to guide every single business process and decision. Missing or incorrect information seriously compromises any decision process downstream. Therefore, a crucial, but tedious task for everyone involved in data processing is to verify the quality of their data. We present a system for automating the verification of data quality at scale, which meets the requirements of production use cases. Our system provides a declarative API, which combines common quality constraints with user-defined validation code, and thereby enables ‘unit tests’ for data. We efficiently execute the resulting constraint validation workload by translating it to aggregation queries on Apache Spark. Our platform supports the incremental validation of data quality on growing datasets, and leverages machine learning, e.g., for enhancing constraint suggestions, for estimating the ‘predictability’ of a column, and for detecting anomalies in historic data quality time series. We discuss our design decisions, describe the resulting system architecture, and present an experimental evaluation on various datasets.

PVLDB Reference Format:

Sebastian Schelter, Dustin Lange, Philipp Schmidt, Meltem Celikel, Felix Biessmann and Andreas Grafberger. Automating Large-Scale Data Quality Verification. *PVLDB*, 11 (12): 1781-1794, 2018.

DOI: <https://doi.org/10.14778/3229863.3229867>

1. INTRODUCTION

Data is at the center of modern enterprises and institutions. Online retailers, for example, rely on data to support customers making buying decisions, to forecast demand [7], to schedule deliveries, and more generally, to guide every single business process and decision. Missing or incorrect information seriously compromises any decision process downstream, ultimately damaging the overall effectiveness and efficiency of the organization. The quality of data has effects

*work done while at Amazon Research

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-nd/4.0/>. For any use beyond those covered by this license, obtain permission by emailing info@vlldb.org.

Proceedings of the VLDB Endowment, Vol. 11, No. 12
Copyright 2018 VLDB Endowment 2150-8097/18/8.
DOI: <https://doi.org/10.14778/3229863.3229867>

across teams and organizational boundaries, especially in large organizations with complex systems that result in complex data dependencies. Furthermore, there is a trend across different industries towards more automation of business processes with machine learning (ML) techniques. These techniques are often highly sensitive on input data, as the deployed models rely on strong assumptions about the shape of their inputs [42], and subtle errors introduced by changes in data can be very hard to detect [34]. At the same time, there is ample evidence that the volume of data available for training is often a decisive factor for a model’s performance [17, 44]. In modern information infrastructures, data lives in many different places (e.g., in relational databases, in ‘data lakes’ on distributed file systems, behind REST APIs, or is constantly being scraped from web resources), and comes in many different formats. Many such data sources do not support integrity constraints and data quality checks, and often there is not even an accompanying schema available, as the data is consumed in a ‘schema-on-read’ manner, where a particular application takes care of the interpretation. Additionally, there is a growing demand for applications consuming semi-structured data such as text, videos and images.

Due to these circumstances, every team and system involved in data processing has to take care of data validation in some way, which often results in tedious and repetitive work. As a concrete example, imagine an on-demand video platform, where the machines that stream videos to users write log files about the platform usage. These log files must regularly be ingested into a central data store to make the data available for further analysis, e.g., as training data for recommender systems. Analogous to all data produced in real-world scenarios, these log files might have data quality issues, e.g., due to bugs in program code or changes in the semantics of attributes. Such issues potentially result in failures of the ingestion process. Even if the ingestion process still works, the errors in the data might cause unexpected errors in downstream systems that consume the data. As a consequence, the team managing the daily ingestion process must validate the data quality of the log files, e.g., by checking for missing values, duplicate records, or anomalies in size.

We therefore postulate that there is a pressing need for increased *automation of data validation*. We present a system that we built for this task and that meets the demands of production use cases. The system is built on the following principles: at the heart of our system is *declarativity*; we want users to spend time on thinking ‘how’ their data should look like, and not have to worry too much about how to im-

plement the actual quality checks. Therefore, our system offers a declarative API that allows users to define checks on their data by composing a huge variety of available constraints. Additionally, data validation tools should provide high *flexibility* to their users. The users should be able to leverage external data and custom code for validation (e.g., call a REST service for some data and write a complex function that compares the result to some statistic computed on the data). Our vision is that users should be able to write ‘unit-tests’ for data (Section 3.1), analogous to established testing practices in software engineering. Furthermore, data validation systems have to acknowledge the fact that *data is being continuously produced*, therefore they should allow for the integration of growing datasets. Our proposed system explicitly supports the incremental computation of quality metrics on growing datasets (Section 3.2), and allows its users to run anomaly detection algorithms on the resulting historical time series of quality metrics (Section 3.4). The last principle to address is *scalability*: data validation systems should seamlessly scale to large datasets. To address this requirement, we designed our system to translate the required data metrics computations to aggregation queries, which can be efficiently executed at scale with a distributed dataflow engine such as *Apache Spark* [50].

The contributions of this paper are the following:

- We present a declarative API combining common quality constraints with user-defined validation code, which enables ‘unit tests’ for data (Section 3.1).
- We discuss how to efficiently execute the constraint validation by translating checks to aggregation queries with dedicated support for incremental computation on growing datasets (Sections 3.2 and 4).
- We give examples of how machine learning can be leveraged in data quality verification, e.g., for enhancing constraint suggestions, for estimating the predictability of a column, and for detecting anomalies in historic data quality timeseries (Sections 3.3 and 3.4).
- We present an experimental evaluation of our proposed approaches (Section 5).

2. DATA QUALITY DIMENSIONS

We first take a look at data quality literature to understand common dimensions of quality. The quality of data can refer to the *extension* of the data (i.e., data values), or to the *intension* of the data (i.e., the schema) [4]. We focus on extensional data quality in the following. We treat schema problems with the concept of completeness; for example, if there is no attribute value specified in the open schema of an entity, we consider it missing. There are multiple studies on measuring the extensional data quality [4, 30, 39]. In the following, we briefly describe the most commonly referred to dimensions.

Completeness refers to the degree to which an entity includes data required to describe a real-world object. In tables in relational database systems, completeness can be measured by the presence of null values, which is usually interpreted as a missing value. In other contexts, for instance in a product catalog, it is important to calculate completeness given the correct context, i.e., the schema of the product category. For example, the absence of a value for the attribute `shoe_size` is not relevant for products in the category `notebooks`. In this case, the attribute value is missing

because the attribute is not applicable [37]. We are only interested in measuring completeness when an attribute is applicable.

Consistency is defined as the degree to which a set of semantic rules are violated. *Intra-relation constraints* define a range of admissible values, such as a specific data type, an interval for a numerical column, or a set of values for a categorical column. They can also involve rules over multiple columns, e.g., ‘if `category` is *t-shirts*, then the range of `size` is {S, M, L}’. *Inter-relation constraints* may involve columns from multiple tables. For example, a column `customerId` may only include values from a given reference table of all customers.

Accuracy is the correctness of the data and can be measured in two dimensions: syntactic and semantic. Syntactic accuracy compares the representation of a value with a corresponding definition domain, whereas semantic accuracy compares a value with its real-world representation. For example, for a product attribute `color`, a value `blue` can be considered syntactically accurate in the given domain even if the correct value would be `red`, whereas a value `XL` would be considered neither semantically nor syntactically accurate.

3. APPROACH

We introduce our general machinery for automated large-scale data quality verification. We first present our declarative API, which allows users to specify constraints on their datasets, and detail how we translate these constraints into computations of metrics on the data, which allow us to subsequently evaluate the constraints (Section 3.1). Next, we discuss how to extend this approach to scenarios where we need to evaluate such constraints for incrementally growing datasets (e.g., in the case of ingestion pipelines in a data warehouse) in Section 3.2. Lastly, we describe extensions of our approach such as the suggestion of constraints (Section 3.3) and anomaly detection on historical data quality time series of a dataset (Section 3.4).

3.1 ‘Unit Tests’ for Data

The general idea behind our system is to enable users to easily define ‘unit tests’ for their datasets in a declarative manner [10]. These ‘unit-tests’ consist of constraints on the data which can be combined with user-defined functions, e.g., custom code. Table 1 shows the constraints available to our users. We want them to focus on the definition of their checks and their validation code, but not on the computation of the metrics required for the constraints. Therefore, we design our system to translate the user-defined constraints into an efficiently executable metrics computation.

Declarative definition of data quality constraints. In our system, users define checks for their datasets, which either result in errors or warnings during execution, if the validation fails. This approach provides a high flexibility for users: they can write complex functions and leverage existing libraries for their validation code; they can use external data or even can call external services. In order to showcase our system, we introduce an exemplary use case on an on-demand video platform. Assume that the machines that stream videos to users write log files about the platform usage, with details such as the type of device used, the length of the session, the customer id or the location.

Table 1: Constraints available for composing user-defined data quality checks.

constraint	arguments	semantic
dimension <i>completeness</i>		
isComplete	column	check that there are no missing values in a column
hasCompleteness	column, udf	custom validation of the fraction of missing values in a column
dimension <i>consistency</i>		
isUnique	column	check that there are no duplicates in a column
hasUniqueness	column, udf	custom validation of the unique value ratio in a column
hasDistinctness	column, udf	custom validation of the unique row ratio in a column
isInRange	column, value range	validation of the fraction of values that are in a valid range
hasConsistentType	column	validation of the largest fraction of values that have the same type
isNonNegative	column	validation whether all values in a numeric column are non-negative
isLessThan	column pair	validation whether values in the 1s column are always less than in the 2nd column
satisfies	predicate	validation whether all rows match predicate
satisfiesIf	predicate pair	validation whether all rows matching 1st predicate also match 2nd predicate
hasPredictability	column, column(s), udf	user-defined validation of the predictability of a column
statistics (can be used to verify dimension <i>consistency</i>)		
hasSize	udf	custom validation of the number of records
hasTypeConsistency	column, udf	custom validation of the maximum fraction of values of the same data type
hasCountDistinct	column	custom validation of the number of distinct non-null values in a column
hasApproxCountDistinct	column, udf	custom validation of the approx. number of distinct non-null values
hasMin	column, udf	custom validation of a column's minimum value
hasMax	column, udf	custom validation of a column's maximum value
hasMean	column, udf	custom validation of a column's mean value
hasStandardDeviation	column, udf	custom validation of a column's standard deviation
hasApproxQuantile	column, quantile, udf	custom validation of a particular quantile of a column (approx.)
hasEntropy	column, udf	custom validation of a column's entropy
hasMutualInformation	column pair, udf	custom validation of a column pair's mutual information
hasHistogramValues	column, udf	custom validation of column histogram
hasCorrelation	column pair, udf	custom validation of a column pair's correlation
time		
hasNoAnomalies	metric, detector	validation of anomalies in time series of metric values

```

1 val numTitles = callRestService(...)
2 val maxExpectedPhoneRatio = computeRatio(...)
3
4 var checks = Array()
5
6 checks += Check(Level.Error)
7 .isComplete("customerId", "title",
8   "impressionStart", "impressionEnd",
9   "deviceType", "priority")
10 .isUnique("customerId", "countryResidence",
11   "deviceType", "title")
12 .hasCountDistinct("title", _ <= numTitles)
13 .hasHistogramValues("deviceType",
14   _.ratio("phone") <= maxExpectedPhoneRatio)
15
16 checks += Check(Level.Error)
17 .isNonNegative("count")
18 .isLessThan("impressionStart", "impressionEnd")
19 .isInRange("priority", ("hi", "lo"))
20
21 checks += Check(Level.Warning, on="delta")
22 .hasNoAnomalies(Size, OnlineNormal(stdDevs=3))
23 checks += Check(Level.Error, on="delta")
24 .hasNoAnomalies(Size, OnlineNormal(stdDevs=4))
25
26 checks += Check(Level.Warning)
27 .hasPredictability("countryResidence",
28   ("zipCode", "cityResidence"), precision=0.99)
29
30 Verification.run(data, checks)

```

Listing 1: Example for declarative data quality constraint definitions using our API.

These log files must regularly be ingested into a central data store to make the data available for further analysis, e.g., as training data for recommender systems. Analogous to all data produced in real-world scenarios, these log files might have data quality issues, e.g., due to bugs in program code, data loss, redeployments of services, or changes in semantics of data columns. Such issues might potentially result in several negative consequences, e.g., the ingestion process might fail and need to be manually restarted after communication with the data provider. Even if the ingestion process still works, the errors in the data might cause unexpected errors in downstream systems that consume the data. In many cases these errors might be hard to detect, e.g., they might cause regressions in the prediction quality of a machine learning model, which makes assumptions about the shape of particular features computed from the input data [34]. Therefore, the video streaming service could use our system to validate the data quality before starting the data ingestion process, by declaring a custom set of checks that should hold on the data. Listing 1 depicts a toy example of how such a declarative quality check for video stream logs could look like and highlights the combination of declarative constraint definitions with custom code. External data is fetched in the beginning and used throughout the quality check: an external REST service is called to determine the overall number of movies in the system and the expected ratio of smartphone watchers is computed (see lines 1 & 2). Then, a set of completeness and consistency checks is defined, e.g., we require the columns `customerId`, `title`, `impressionStart`,

Table 2: Computable metrics to base constraints on.

metric	semantic
dimension <i>completeness</i>	
Completeness	fraction of non-missing values in a column
dimension <i>consistency</i>	
Size	number of records
Compliance	ratio of columns matching predicate
Uniqueness	unique value ratio in a column
Distinctness	unique row ratio in a column
ValueRange	value range verification for a column
DataType	data type inference for a column
Predictability	predictability of values in a column
statistics (can be used to verify dimension <i>consistency</i>)	
Minimum	minimal value in a column
Maximum	maximal value in a column
Mean	mean value in a column
StandardDeviation	standard deviation of the value distribution in a column
CountDistinct	number of distinct values in a column
ApproxCountDistinct	number of distinct values in a column estimated by a hyperloglog sketch [21]
ApproxQuantile	approximate quantile of the value in a column [15]
Correlation	correlation between two columns
Entropy	entropy of the value distribution in a column
Histogram	histogram of an optionally binned column
MutualInformation	mutual information between two columns

impressionEnd, **deviceType** and **priority** to be complete (lines 7 to 9), and we dictate that the column combination **customerId**, **countryResidence**, **deviceType**, and **title** is unique in the data at hand (lines 10 and 11). We make sure that the number of distinct values in the **title** column is less than or equal to the overall number of movies in the system (line 12) and we check that the ratio of ‘phone’ devices meets our expectations by investigating a histogram of the **deviceType** column in lines 13 and 14. Subsequently, we issue another set of consistency checks that define the expected shape of the data (e.g., no negative values in the **count** column, a happens-before relationship between the viewing timestamps, and a set of valid values for the **priority** column, lines 16 to 19).

Next, we have two checks that rely on comparisons to previously computed metrics on former versions of the dataset (available from a central ‘metrics database’): we advise the system to detect anomalies in the time series of sizes of records that have been added to the dataset over time and issue a warning if the size is more than three standard deviations away from the previous mean and throw an error if it is more than four standard deviations away (see lines 21 to 24). Finally, we define a predictability check for the **countryResidence** column which dictates that our system should be able to predict values in this column with a precision of 99% by inspecting the corresponding values in the **zipCode** and **cityResidence** columns.

Translating constraints to metrics computations. In the following, we detail how our system executes the actual data quality verification. The declarative definition of constraints (which are evaluated by the user code) re-

lies on a particular set of data quality metrics that our system computes from the data at hand. The system inspects the checks and their constraints, and collects the metrics required to evaluate the checks. Table 2 lists all data quality metrics supported by our system. We directly address the data quality dimensions *completeness* and *consistency* listed in Section 2. Let D denote the dataset D with N records, on which we operate, and let c_v denote the cardinality of value v in a particular column of dataset D . Furthermore, let V denote the set of unique values in a particular column of the dataset D . We calculate **Completeness** as the fraction of non-missing values in a column: $|\{d \in D \mid d(\text{col}) \neq \text{null}\}| / N$. For measuring *consistency*, we provide metrics on the number of unique values, the data types, the data set size, the value ranges, and a general predicate matching metric. The **Size** metric for example refers to the number of records N , while the **Compliance** metric denotes the ratio of records which satisfy a particular predicate: $|\{d \in D \mid p(d)\}| / N$. The metric **Uniqueness** refers to the unique value ratio [19] in a particular column: $|\{v \in V \mid c_v = 1\}| / |V|$, while **Distinctness** corresponds to the unique row ratio $|V| / N$ in the column. In addition, we implement standard summary statistics for numerical columns that can be used for defining additional semantic rules on datasets, such as **Minimum**, **Maximum**, **Mean**, **StandardDeviation**, **Histogram**, and **Entropy**, which we for example compute as $-\sum_v \frac{c_v}{N} \log \frac{c_v}{N}$. We also include standard statistics such as **Correlation** and **MutualInformation** for measuring the amount of association between two columns, where the latter is computed as: $\sum_{v_1} \sum_{v_2} \frac{c_{v_1} c_{v_2}}{N} \log \frac{c_{v_1} c_{v_2}}{c_{v_1} c_{v_2}}$. As some metrics are rather expensive to compute and might involve re-partitioning or sorting the data, our system provides approximations of metrics such as quantiles in the form of **ApproxQuantile** (computed via an efficient online algorithm [15]) or **ApproxCountDistinct** for estimating the number of distinct values with a hyperloglog sketch [21].

Lastly, we offer an implementation of **Predictability**. In an attempt to automate the verification of the correctness of values, we train a machine learning model that predicts a value for a target column t of a particular record from all k observed values $l_1, \dots, l_k \in V_t$ in the target column, given the corresponding values l_{i_1}, \dots, l_{i_n} of input columns i_1, \dots, i_n for the particular record, e.g., using the maximum a posteriori decision rule: $\text{argmax}_k p(l_k | l_{i_1}, \dots, l_{i_n})$. An example would be to predict the value of a ‘color’ column in a product table from text in the ‘description’ and ‘name’ columns. We train this model on a sample of observed values in the target column, and measure its prediction quality on the held-out rest of the data. We return the quality score, calculated using standard measures such as precision, recall or F_1 -score of the predictions, as value of the metric.

After having inspected the checks and collected the metrics, the system triggers the efficient computation of the metrics (see Section 4 for details on how we physically execute these computations), invokes the user-defined validation code from the constraints, and evaluates the results.

Output. After execution of the data quality verification, our system reports which constraints succeeded and which failed, including information on the predicate applied to the metric into which the constraint was translated, and the value that made a constraint fail.

```

...
Success("isComplete(title)",
    Completeness("title") == 1.0)),
Success("isNonNegative(count)",
    Compliance("count >= 0") == 1.0),
Failure("isUnique(customerId, countryResidence,
    deviceType, title"),
    Uniqueness("customerId", "countryResidence",
    "deviceType", "title") == 1.0, 0.9967),
Failure("isInRange(priority, ('hi', 'lo'))",
    Compliance("priority IN ('hi', 'lo')") == 1.0,
    0.833),
...

```

Listing 2: Exemplary output of data quality verification showing metrics, applied predicates and results.

Listing 2 shows an excerpt of a potential output for our example. We see that our `isComplete(title)` constraint has been translated to a predicate `Completeness(title) == 1.0` which held on the data. Analogously, our constraint `isNonNegative(count)` leads to the predicate `Compliance("count >= 0") == 1.0` and also matched all records. On the contrary, our unique constraint has failed, as only 99.67% of records have been identified as unique, and the predicate which the system generated from the `isInRange` constraint only matched 83.3% of records.

3.2 Incremental Computation of Metrics for Growing Datasets

In real-world deployments, data is seldomly static; instead we typically encounter systems that continuously produce data (e.g., by interacting with users). Therefore it is of utter importance for data validation systems like ours to support scenarios where we continuously ingest new batches of records for a particular dataset. In such cases, we need access to updated metrics for the whole dataset as well as for the new records and we must be able to update such metrics incrementally without having to access the previous data (see Figure 1 for details). In the following, we present our incremental metrics computation machinery built for this task.

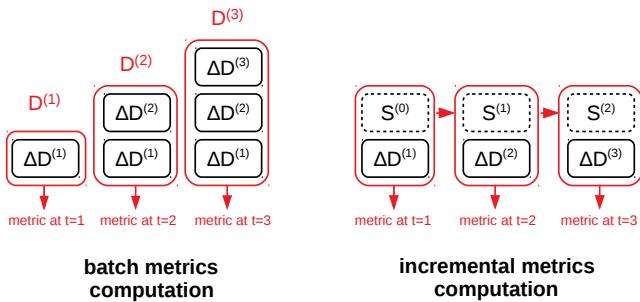


Figure 1: Instead of repeatedly running the batch computation on growing input data D , we support running an incremental computation that only needs to consume the latest dataset delta $\Delta D^{(t)}$ and a state S of the computation.

Computational model. Let $D^{(t)}$ denote the snapshot of dataset at time t and let $\Delta D^{(t)}$ denote the delta data at time t (the additional records) required to form $D^{(t+1)}$.

Note that we restrict ourselves to append-only cases where a dataset at time t is simply the union of all previous deltas: $D^{(t)} = \bigcup_{k=1}^t \Delta D^{(k)}$. Instead of computing metrics for the growing dataset from all snapshots, incremental computation introduces a state S , a function f for updating the state from a delta and the previous state, and a function g for computing the actual metric from the state S such that $m^{(t)} = g(S^{(t)})$. Furthermore, we need an initial ‘empty’ state $S^{(0)}$. The benefit of incremental computation is that it allows us to compute the series of metrics for the dataset snapshots via a recursive computation that *only consumes the deltas*:

$$S^{(t)} = f(\Delta D^{(t)}, S^{(t-1)})$$

Reformulation of quality metrics. In the following, we present a set of reformulations of the existing metrics to enable incremental computation for them. For each metric, we show how to ‘split’ the computation of the metrics for the new dataset $D^{(t+1)}$ into the computation of sufficient statistics over the previous dataset D and the dataset delta ΔD (we drop the indexes here to improve readability). Once such a reformulation is given, we can conduct the computation for $D^{(t+1)}$ by loading the persisted sufficient statistics for D and updating these from values computed only on the newly added records ΔD .

Notation: Let N and ΔN denote the number of records in the datasets D and ΔD . Let V and ΔV denote all unique values in a particular column of the dataset D or ΔD . The set of unique values in the new dataset $V^{(t+1)}$ is simply the union $V \cup \Delta V$ of the sets of unique values from the previous dataset and the delta dataset. Furthermore, let c_v and Δc_v denote the cardinality of value v in a particular column of dataset D or ΔD .

The number of records **Size** is the most straightforward metric to rewrite, as the size of the new dataset $D^{(t+1)}$ is simply the sum $N + \Delta N$ of the size N of the previous dataset D plus the size ΔN of the delta dataset ΔD . For an incremental version of **Compliance**, we need to maintain two intermediate results, namely the absolute number of records $|\{d \in D \mid p(d)\}|$ that previously matched the predicate as well as the size N of the previous dataset D . Then we can compute the compliance for the new dataset $D^{(t+1)}$ from these retained values and the number of records $|\{d \in \Delta D \mid p(d)\}|$ that matched the predicate in the delta as well as the size ΔN of the delta:

$$\frac{|\{d \in D \mid p(d)\}| + |\{d \in \Delta D \mid p(d)\}|}{N + \Delta N}$$

We can reformulate **Completeness** as compliance with an ‘is not null’ predicate. The incremental computation of **Uniqueness** requires us to know the cardinalities c_v of the value v in the previous dataset D as well as the set of distinct values V . We need to inspect the sum of the cardinalities $c_v + \Delta c_v$ for each value v in the previous dataset and the delta:

$$\frac{|\{v \in V \cup \Delta V \mid c_v + \Delta c_v = 1\}|}{|V \cup \Delta V|}$$

We also compute incremental **Distinctness** along these lines by comparing the number of distinct values in the data $|V \cup \Delta V|$ to the size of the data $N + \Delta N$:

$$\frac{|V \cup \Delta V|}{N + \Delta N}$$

Incremental computation of **Entropy** requires us to estimate the probability $p(v)$ of a particular value v occurring in the column from the value's cardinality c_v in the previous data, its cardinality Δc_v in the delta and the sizes N and ΔN of the previous dataset and the delta:

$$-\sum_v \frac{c_v + \Delta c_v}{N + \Delta N} \log \frac{c_v + \Delta c_v}{N + \Delta N}$$

The incremental computation of **MutualInformation** requires us to maintain histograms about the cardinalities c_{v_1} of the first column, c_{v_2} of the second column, as well as cooccurrence counts $c_{v_1 v_2}$ for all pairwise occurrences, and merge these with the corresponding counts $\Delta c_{v_1 v_2}$ for the delta dataset:

$$\sum_{v_1} \sum_{v_2} \frac{c_{v_1 v_2} + \Delta c_{v_1 v_2}}{N + \Delta N} \log \frac{c_{v_1 v_2} + \Delta c_{v_1 v_2}}{(c_{v_1} + \Delta c_{v_1})(c_{v_2} + \Delta c_{v_2})}$$

In order to compute our **Predictability** metric, we evaluate the prediction quality of a multinomial naive bayes model (trained on features extracted from the user-specified input columns) for the target column. The parameters are typically estimated using a smoothed version of the maximum likelihood estimate: $\text{argmax}_k \sum_i f_i \log \frac{N_{ki} + \alpha_i}{N_k + \alpha}$. Here, N_{ki} denotes the number of times feature i occurs in class k , N_k stands for the overall number of feature occurrences in class k , a uniform prior used for the sake of simplicity, α_i is the smoothing term per feature and α the sum of the smoothing terms. For updating a classification model from a previous dataset D to $D^{(t+1)}$, we need to know $N_{ki}^{(t+1)}$ and $N_k^{(t+1)}$, which we can easily compute by adding the counts ΔN_{ki} and ΔN_k from the delta ΔD to the counts N_{ki} and N_k for the previous version $D^{(t)}$ of the dataset:

$$\text{argmax}_k \sum_i f_i \log \frac{N_{ki} + \Delta N_{ki} + \alpha_i}{N_k + \Delta N_k + \alpha}$$

The data structures which we use for the **ApproxQuantile** and **ApproxCountDistinct** metrics naturally support incremental computations and therefore do not require special care from our side.

3.3 Constraint Suggestion

The benefits of our system to users heavily depend on the richness and specificity of the checks and constraints, which the users define and for which our system will regularly compute data quality metrics. As a consequence, it is very important for a system like ours to make the adoption process as simple as possible. Therefore we provide machinery to automatically suggest constraints and identify data types for datasets (even if no schema is available). Such suggestion functionality can then be integrated into ingestion pipelines and can also be used during exploratory data analysis. The starting point for our constraint suggestion machinery is a dataset where individual columns are known and have names, but no further schema information such as data types or constraints is available. A classical example for such a dataset would be a CSV file living in a distributed filesystem. Our system assists the user in identifying data types of columns and suggests potential constraints to users, which they can use as a foundation to design declarative checks for the dataset at hand.

Heuristics on summary statistics. Our constraint suggestion functionality is built on a heuristics-based approach employing single-column profiling [1]. While more complex data profiling would certainly be helpful, we are required to be able to consume terabyte-sized tables with several billions of rows, and therefore have to restrict ourselves to simple statistics. As already explained, the user provides a single table dataset with no type information and no schema information except column names as input. Furthermore, the user can optionally specify a set of columns to inspect (and a sample size to use during the suggestion phase) to speedup the process. Our system then executes single column profiling in three passes over the data. In the first pass, we compute the data size, run data type detection on each column, and compute the completeness as well as the approximate number of distinct values via hyperloglog sketches [21, 18] for each column of interest. The profiling tasks in the second pass operate on the columns which we identified to have numeric types. For every such column, we compute summary statistics such as the minimum, maximum, mean, standard deviation, and approximate quartiles [15]. In a third pass, we compute the frequency distribution of values for columns with a cardinality below a user-specified threshold (in order to bound the required memory). Afterwards, our system recommends constraints for the dataset at hand, based on heuristics leveraging the profiling results. In the following, we list a selection of the heuristic rules which we apply:

- If a column is complete in the sample at hand, we suggest an **isComplete** (not null) constraint.
- If a column is incomplete in the sample at hand, we suggest a **hasCompleteness** constraint. We model the fact whether a value is present or not as a Bernoulli-distributed random variable, estimate a confidence interval for the corresponding probability, and return the start value of the interval as lower bound for the completeness in the data.
- If the detected type of the column is different from ‘string’, we suggest a **hasConsistentType** constraint for the detected type.
- For key discovery, we investigate an approximation of the ‘unique row ratio’ [12]: if the ratio of dataset size to the approximated number of distinct values in that column is within the approximation error bound of the used hyperloglog sketch, we suggest an **isUnique** constraint.
- If a column is numeric and its observed values fall into a certain range, we suggest a **Compliance** constraint with a predicate that matches only values within that range (e.g., a range of only positive values if the observed minimum is 0).
- If the number of distinct values in a column is below a particular threshold, we interpret the column as categorical and suggest an **isInRange** constraint that checks whether future values are contained in the set of already observed values.

Note that we see constraint suggestion as a ‘human-in-the-loop’ process with a low computational budget and therefore rely on the end user to select from and validate our suggestions which might not necessarily hold for future data (or even the sample at hand in the case of unique constraint suggestions).

Learning semantics of column and table names. We notice that the actual names of columns often contain inherent semantics that allow humans to intuitively infer column types and constraints. Examples for such names are ‘id’, which is commonly used for an artificial primary key column of type string or int, ‘is_deleted’, which probably refers to boolean column, or ‘price_per_unit’ which indicates a numeric column. We therefore train a machine learning model to predict constraints solemnly based on the name of the table and column, as well as its type. The training data for this model is extracted from the schemas of tables from open source projects. Our system integrates this model by leveraging its predictions to enhance (and potentially correct) the suggestions made by our heuristics. In the case where our heuristic rules suggest an `isUnique` constraint, we consult the classifier’s probabilistic prediction to decide whether to follow the suggestion or not.

3.4 Anomaly Detection

Anomaly detection in our system operates on historic time series of data quality metrics (e.g., the ratio of missing values for different versions of a dataset). We pose no restriction on the anomaly detection algorithm to apply and our system ships with a handful of standard algorithms. Examples are an algorithm that simply checks for user-defined thresholds. The algorithm from our example called `OnlineNormal` computes a running mean and variance estimate and compares the series values to a user-defined bound on the number of standard deviations they are allowed to be different from the mean. An additional method allows users to specify the degree of differencing applied prior to running the anomaly detection. This gives users the possibility to apply a simple technique in order to stationarize the to-be analysed time series [22]. Data quality metrics such as the number of missing values in a continuously produced dataset might be subject to seasonality or trends (e.g., the loss only occurs at certain times when the system is under heavy load). In these cases, asserting the correct behaviour may not be feasible with user-supplied thresholds. To this end, we allow users to plug in their own algorithms for anomaly detection and time series prediction.

4. IMPLEMENTATION

We implement our data validation library on top of the distributed dataflow engine *Apache Spark* [50], using AWS infrastructure for storage. Note that our library does not depend on any functionality exclusive to Spark, and would be easily extendable to leverage different runtimes, as long as they support SQL queries, user-defined aggregation functions and simple machine learning models¹. We decided for Spark due to the fact that a Scala/JVM environment makes it very easy for users to write custom verification code and interact with external libraries and systems. Figure 2 gives an overview of the applied architecture. Our system operates on `DataFrames`, a relational abstraction for a partitioned (and often denormalized) table. The user-facing API consists of so-called `Checks` and `Constraints`, which allow our users to declaratively define on which statistics of the data their verification code should be run. When executing the checks, our library inspects the contained constraints

¹A system with support for materialized views would even allow us to simplify our incremental computation machinery.

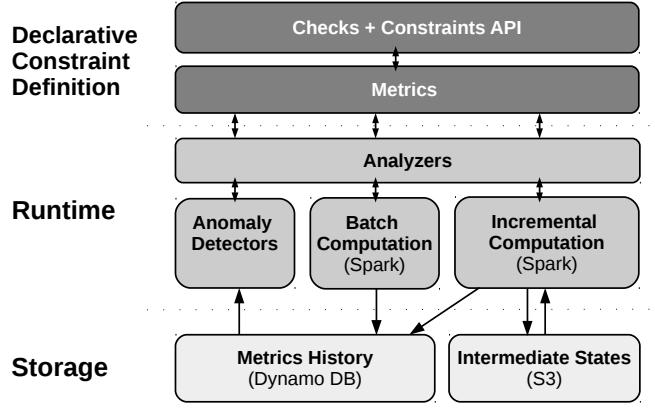


Figure 2: System architecture: users declaratively define checks and constraints to combine with their verification code. The system identifies the required metrics and efficiently computes them in the runtime layer. The history of metrics and intermediate states of incremental computations are maintained in AWS storage services.

and identifies the required `Metrics` that must be computed on the data in order to run the user-defined verification code of the constraints. For each metric, our library chooses a so-called `Analyzer` (which can be seen as a physical operator) capable of computing the particular metric on the data. The selected `Analyzers` are given to an `AnalysisRunner` in our runtime layer which schedules the execution of the metrics computation. This runner applies a set of simple optimizations to the computation of multiple metrics. For all metrics that do not require re-partitioning the data, the runner collects their required aggregation functions and executes them in a single generated `SparkSQL` query over the data to benefit from scan-sharing. In our example from Section 3.1, such metrics would be the `Size` of the dataset, the `Completeness` of six columns, as well as the `Compliance` for the three `satisfies` constraints. All these metrics will be computed simultaneously in a single pass over the data. The resulting metrics are finally stored in a document database (DynamoDB) for later retrieval (and usage by anomaly detection algorithms). The runtime for incremental computations stores the states of incremental analyzers in a distributed filesystem (S3).

For the predictability estimation, we have to train a machine learning model on the user-specified input columns and evaluate how well it predicts values in the target column. We developed a pluggable architecture, where we featurize the input columns by concatenating their string representations, and tokenize and hash them via Spark’s `Tokenizer` and `HashingTF` transformers. Afterwards, any classification algorithm from `SparkML` [27] can be used to learn a prediction model; in our experiments we used Sparks Naive Bayes [36] implementation as it offers a scalable lower bound on prediction accuracy, does not require hyperparameter optimization, and is simple to train incrementally. We apply the trained classification model to predict values on a held-out fraction of the data and report the prediction quality (e.g., measured using precision) as predictability value.

4.1 Incremental Computation

In the following, we detail how to make our system’s analyzers ‘state-aware’ to enable them to conduct incremental computations. A corresponding base class in Scala is shown in Listing 3, where M denotes the type of metric to compute and S denotes the type of state required. Persistence and retrieval of the state are handled outside of the implementation by a so-called `StateProvider`. The method `initialState` produces an initial empty state, `apply` produces a state and the corresponding metric for an initial dataset, and `update` consumes the current state and a delta dataset, and produces the updated state, as well as the corresponding metrics, both for the dataset as a whole and for the delta, in the form of a tuple (S, M, M) . Furthermore, the method `applyOrUpdateFromPersistedState` executes the incremental computation and takes care of managing the involved states using `StateProviders`.

```

1 trait IncrementalAnalyzer[M, S]
2   extends Analyzer[M] {
3
4   def initialState(initialData: DataFrame): S
5
6   def update(
7     state: S,
8     delta: DataFrame): (S, M, M)
9
10  def updateFromPersistedState(
11    stateProvider: Option[StateProvider],
12    nextStateProvider: StateProvider,
13    delta: DataFrame): (M, M)
14 }
15
16 trait StateProvider {
17
18   def persistState[S](
19     state: S,
20     analyzer: IncrementalAnalyzer[M, S])
21
22   def loadState[S](
23     analyzer: IncrementalAnalyzer[M, S]): S
24 }
```

Listing 3: Interface for incremental analyzers.

State management. In order to execute the incremental computation, a user needs to configure state providers to allow for state management. Typically, the state for a particular snapshot of the dataset resides in a directory on S3 and we provide a corresponding provider implementation. Given these, we call `applyOrUpdateFromPersistedState` which will compute the metric and persist the state. To compute the updated metric for the next snapshot of the dataset, we need two `StateProviders`, one which provides the state for the old snapshot, and another one which will receive the updated state computed from the old state and the delta. Note that this internal API is typically hidden from the users, which are advised to program our system using the declarative checks API from Section 3.1. In the following we discuss additional implementation details. When incrementally computing metrics that require a re-partitioning of the data (e.g., entropy and uniqueness that require us to group the data by the respective column), we implement the incremental scheme as follows. The `state S` is composed of a histogram over the data (the result of `delta.select(columns).groupBy(columns).count()`). The `update function` merges

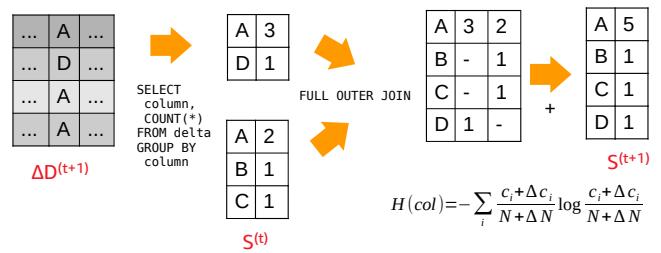


Figure 3: Example for an incremental update of the entropy of a column: the frequencies of values in the delta records $\Delta D^{(t+1)}$ are computed via a grouping query and merged with the previous state $S^{(t)}$ via a full outer join. After adding up the counts, we have the updated state $S^{(t+1)}$, from which the entropy of the column in the updated dataset $D^{(t+1)}$ can be computed.

the previous histogram for the current data with the histogram of the delta via an outer join on the grouping columns and computes the corresponding counts for the delta and the whole dataset. The analyzer then computes the metric from the state by an aggregation over the histogram. We showcase an example of incrementally updating the entropy of column in Figure 3.

Optimizations. During the computation of multiple metrics, we apply a set of manually enforced query optimizations: (a) we cache the result of the `count` operation on dataframes, as many metrics require the size of the delta for example; (b) we apply `scan sharing` for aggregations: we run all aggregations that rely on the same grouping (or no grouping) of the data in the same pass over the data.

4.2 Efficiently Suggesting Constraints

The major design objective in our constraint suggestion component is to keep the computation of required summary statistics cheap so that it can be executed during an ingestion pipeline for large datasets. It is therefore crucial to keep the number of passes over the data independent of the number of columns in the dataframe at hand. We assume our input to be a dataframe with named columns with unknown types (initially of type ‘string’ during ingestion, e.g., when reading CSV files). For the computation of the summary statistics required for our heuristics from Section 3.3, we only use aggregations that do not require a re-partitioning of the table, and we only conduct two passes over the data, where the aggregations share scans. Furthermore, we have an estimate of the number of distinct values per column of interesting columns after the first pass, which allows us to control the memory for sketches and histograms (e.g., by only computing the full value distribution for columns with low cardinality) used in the second pass.

As discussed in Section 3.3, we leverage a machine learning model for deciding upon unique constraint suggestion. This model’s inputs are the table name, as well as column name and type. As training data for this model, we extract a dataset of 2,453 (`table_name`, `column_name`, `type`, `is_unique`) tuples from the database schemas of several open source projects such as `mediawiki`, `wordpress` and `oscommerce`. On this schema data, we train a logistic regression

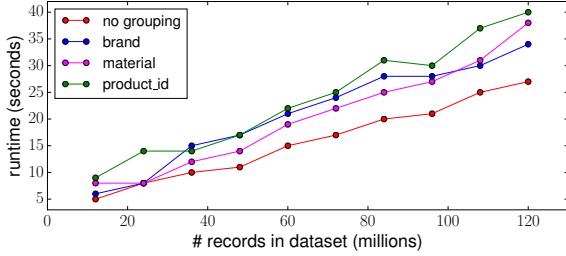


Figure 4: Linearly increasing runtime for different batch metrics computations on a growing product dataset with up to 120 million records.

model using hashed character n-grams of the names and a one-hot-encoding of the type as features. We leverage the `SGDCclassifier` combined with the `HashingVectorizer` from *scikit-learn* [32], and tune the model’s hyperparameters (feature vector dimensionality, regularization, size of n-grams) using 5-fold cross validation. We achieve an AUC score of 0.859 for the ROC curve, using a logistic loss function with L1 regularization and a regularization factor of 0.001 on n-grams of up to size 5 from the input data hashed to 10^8 dimensional feature vectors. We leverage the probabilistic prediction of this model (giving us a hint on whether the naming of the column indicates a unique constraint) as a score for our rule-based unique constraint suggestion and only issue the suggestion to the user if the model assigns a probability greater than 50%.

5. EXPERIMENTAL EVALUATION

In the following, we run a set of scalability experiments for our batch metrics computation, apply our predictability estimation to a product dataset (Section 5.1), and investigate the benefits of applying incremental computation to growing datasets (Section 5.2). Finally, we evaluate our constraint suggestion functionality on two external datasets (Section 5.3) and showcase a simple anomaly detection use-case in Section 5.4.

For our Spark-based experiments, we leverage a dataset representing a sample from an internal product catalog, which consists of approximately 120 million records, where each record describes a product with several hundred attributes; the data size is roughly 50GB in parquet format. We mimic a use case with a growing append-only dataset, and randomly partition our product data into 10 ‘deltas’ of about 12 million records for that. Additionally, we use two external datasets for evaluation. The first dataset² consists of comments from May 2015 in several discussion boards on the social news aggregation website *reddit.com*. The second dataset contains information about 5,180 twitter users, which we extracted from the publicly available twitter sample stream.

The Spark-based experiments leverage a cluster on Elastic MapReduce with 5 workers (c3.4xlarge instances) running Apache Spark 2.0.2 and HDFS 2.7.3.

²<https://www.kaggle.com/reddit/reddit-comments-may-2015>

5.1 Batch Computation

In this first set of experiments, we evaluate how well our metrics computation scales to large datasets and showcase the efficiency of our machine learning-based predictability estimation.

Scalability of metrics computations. In order to evaluate the scalability of our system’s batch metrics computation, we compute a set of quality metrics on the resulting growing product dataset, which we read from S3. Figure 4 shows the results, where each point represents the runtime on a particular version of the growing dataset. The plot labeled ‘no grouping’ refers to the results for computing a set of six metrics (size of the data and completeness of five columns) which do not require us to re-partition the data. Therefore these metrics can be computed by aggregations in a single pass over the data. The remaining lines refer to the computation of metrics such as entropy and uniqueness on the columns `brand`, `material` and `product.id`, which require us to repartition the data (e.g., grouping it by the column for which we want to compute these metrics). Due to the inherent re-partitioning, these metrics are typically more costly to compute and their cost is related to the cardinality of the respective column. Nevertheless, all four evaluated workloads exhibit a runtime linearly growing with the dataset size, which is expected as our system internally generates simple aggregation queries with custom aggregation functions to be run by *SparkSQL* [3].

Predictability estimation with naive bayes. We showcase our predictability estimation functionality on a set of 845,000 fashion products from 10 popular brands which we extracted from the larger product dataset. We set the task of predicting the value of the `brand` column from other columns, such as `name`, `description`, `size`, `bulletpoints`, `manufacturer` and combinations of those. We run the corresponding experiments with Spark on a single c4.8xlarge instance. We take different samples from the dataset (100k records, 200k records, 400k records, 600k records, full dataset). On each of these, we train a naive bayes model with hashed input columns as features for predicting the `brand` column on 80% of the sample. Finally, we calculate the weighted F_1 score for the predictions on the remaining 20%. We repeat this for different input columns and differently sized samples of the data. We find that the `name` column alone is already a very good predictor for `brand`, as it results in a weighted F_1 score of over 97% on all samples of the dataset. The best results are achieved when leveraging a combination of the `name` column with the `bulletpoints`, `description` and `manufacturer` columns, where we reach F_1 scores of 97.3%, 98.8%, 99.4%, 99.5% for the 100k, 200k, 400k, 600k samples of the data, and of 99.5% for the full dataset. Based on these results (which can be computed with an `AnalysisRunner` from Section 4), a user could configure a constraint for future data to get notified once the predictability drops below the observed values, e.g.:

```
Check(Level.Warning)
  .hasPredictability("brand", ("name",
    "bulletpoints", "description",
    "manufacturer"), f1=0.97)
```

We additionally record the runtime of the model training for different featurizations. We find that the runtime scales linearly for growing data and mostly depends on the length

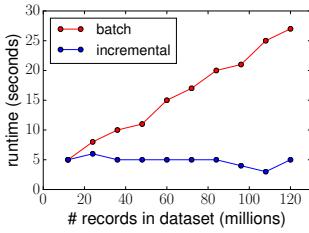


Figure 5: Runtimes for size and completeness on `product_id`, `material`, `color`, `brand`.

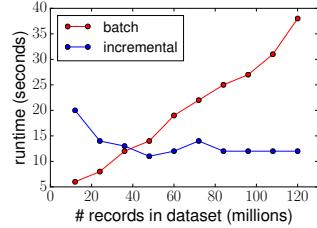


Figure 6: Runtimes for uniqueness and entropy on `material`.

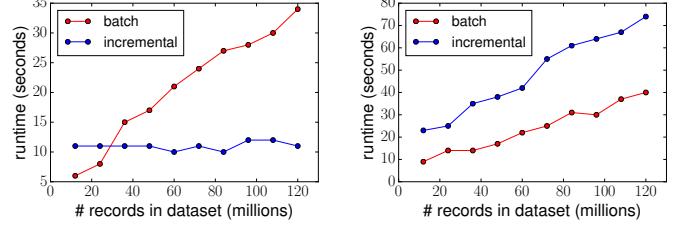


Figure 7: Runtimes for uniqueness and entropy on `brand`.

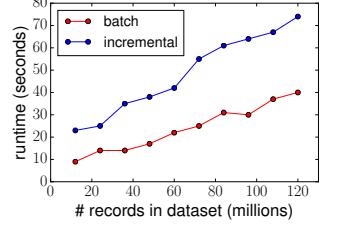


Figure 8: Runtimes for uniqueness and entropy on `product_id`.

of the strings in the input columns (e.g., training a model on the `description` column alone with lots of text takes longer than training on the `name` column combined with the `bulletpoints` column). This is expected as naive bayes conducts a single pass through the data, and sums up the feature vectors per class, which result from tokenizing and hashing the input columns. Note that the training is very efficient; it takes less than ten seconds in all cases, even on the full dataset.

5.2 Benefits of Incremental Computation

We revisit our scalability experiment on growing data to validate our assumptions about the benefits of incremental computation. We compare batch analysis, which always has to consume the dataset as a whole (the union of all the deltas seen so far) against our incremental approach which maintains a state and always operates on this state and the current delta only.

Figure 5 shows the results for again computing the metrics that do not require us to re-partition the data (we referred to this experiment as ‘non-grouping’ previously). These metrics can be computed in a single pass over the data, and the actual state for the incremental computation is tiny here, as only one or two numbers per metric have to be maintained. While the runtime of the batch analysis grows linearly with the dataset size, the runtime remains constant in the incremental case, as it only depends on the size of the delta (which is constant in our setup). Next, we revisit the computation of metrics such as entropy and uniqueness which require us to repartition the data. These metrics are typically more costly to compute and the incremental computation is also more difficult, as we have to internally maintain a histogram of the frequencies per value in the column (the result of the grouping operation). We first compute these metrics on the columns `material` and `brand` which have a rather low cardinality. The results are shown in Figure 6 and Figure 7. We see that the incremental approach has a substantive overhead in this case (persisting and joining the maintained histogram), however its runtime stays roughly constant and it outperforms the batch analysis after three or four deltas. Figure 8 shows the resulting runtimes for computing entropy and uniqueness for the `product_id` column. This column is special in this dataset as it consists of unique values only. Due to this characteristic, the runtime of the incremental approach shows the same growth behavior as the runtime of the batch analysis (linearly growing with data size), as every delta introduces a set of new values, and the histogram which the incremental computation maintains

is basically just a copy of the original column. The overhead of maintaining this histogram is also what makes the incremental computation always perform worse. While this is a drawback, the incremental approach still has the advantage of not requiring access to the full dataset during computation time, which greatly simplifies ingestion pipelines.

5.3 Constraint Suggestion

We evaluate our constraint suggestion component on a sample of 50,000 records from the reddit dataset as well as on the twitter users dataset. In each case, we take a random sample of 10% of the records, have our system suggest constraints based on the sampled data and compute the coverage of these constraints on the remaining 90% of the datasets. The suggested constraints as well as their coverage is shown in Table 3. The reddit dataset is a very easy case, as all columns are complete and only have types string and integer. This simple structure is reflected by the fact that all suggested constraints suggested hold on the test set. The experiment on the twitter users dataset is more interesting, as we have columns with missing values such as `location` and columns with a small set of discrete values such as `lang`. The completeness of the `location` column in the sample is 0.28076 and the suggested constraint `hasCompleteness >= 0.28` holds on the test data, e.g., the ratio of missing values does not increase. The system correctly suggests an `isUnique` constraint for the columns `id` and `screen_name` both of which are actually primary keys for the data. However, the system also suggests two constraints which do not hold for the data. The first one is the range of values for the `lang` column. Here we identified ten different values which only account for more than 99% of records in the test data, but miss rare languages such as turkish or hungarian. A failing constraint on that column can nevertheless be helpful; we found by manual investigation that the data in this column is not correctly normalized, e.g., there are different capitalizations of the same language value such as ‘en-gb’ and ‘en-GB’. In the second case, the system erroneously suggests an `isUnique` constraint for the `statuses_count` column, due to the fact that there are many different values for this column in the sample at hand and we only known an approximation of the number of distinct values; the uniqueness value of this column is only 64% percent in the test data. The second error is corrected, however, once we leverage the predictions of our classifier for unique constraints: while the classifier assigns a high probability of 81% that our suggested unique constraint on the `id` column is valid, it only assigns a 2% probability to the

Table 3: Constraint suggestion and type prediction for the reddit comments and twitter users dataset. Constraints are suggested based on a 10% sample of the data, and their coverage is computed on the remaining 90% of the data. We leverage a machine learning model trained on column names to decide upon potential unique constraints.

dataset	column	suggested constraints	coverage	classifier score
reddit-comments	id	isComplete, isUnique	1.0	-
	created_utc	isComplete, hasConsistentType(integral), isNonNegative	1.0	0.83
	subreddit	isComplete	1.0	-
	author	isComplete	1.0	-
	ups	isComplete, hasConsistentType(integral)	1.0	-
	downs	isComplete, hasConsistentType(integral), isNonNegative	1.0	-
	score	isComplete, hasConsistentType(integral)	1.0	-
	edited	isComplete, isNonNegative	1.0	-
	controversiality	isComplete, hasConsistentType(integral), isNonNegative, isInRange(0, 1)	1.0	-
twitter-users	text	isComplete	1.0	-
	id	isComplete, hasConsistentType(integral), isNonNegative, isUnique	1.0	-
	screen_name	isComplete, isUnique	1.0	0.81
	lang	isComplete, isInRange('en', 'pt', ...)	1.0	0.01
	location	hasCompleteness >= 0.28	0.991	-
	followers_count	isComplete, hasConsistentType(integral), isNonNegative	1.0	-
	statuses_count	isComplete, hasConsistentType(integral), isNonNegative, isUnique	1.0	-
	verified	isComplete, hasConsistentType(boolean)	0.636	0.02
	geo_enabled	isComplete, hasConsistentType(boolean)	1.0	-

`statuses_count` column being unique; therefore, we decide against the suggested constraint. Unfortunately, the classifier produces a false negative for the `screen_name` column, which is indeed unique for our sample at hand, however this would also be not immediately obvious to humans (as different users are allowed to have the same screen name on many social networking platforms), and we prefer conservative and robust suggestions (e.g., rather having false negatives than false positives), which build trust in our users.

5.4 Anomaly Detection

In order to showcase our anomaly detection functionality, we apply it to a fictitious use case on the reddit dataset. Assume that we want to leverage the discussion data for an information extraction task such as question answering. A potential data quality problem would now be that the data could contain large amounts of spam and trolling posts, which would negatively influence our machine learning model that we aim to train on the data. If we regularly ingest data from reddit, we would like to be alarmed if there are signs of increased spamming or trolling activity. The reddit data contains a `controversiality` field for each post, and the series of the ratio of such posts in a board per day might be a good signal for detecting potential trolling. To leverage our anomaly detection functionality for this task, we inspect the historic time series of the `Mean(controversiality)` metric per discussion board (`subreddit`) which we would need to compute during ingestions. In our declarative API, the corresponding check looks as follows:

```
Check(Level.Warning, groupBy="subreddit")
  .hasNoAnomalies("controversiality", Mean,
    OnlineNormal(upperDeviationFactor=3))
```

This indicates that we want to be warned if the mean controversiality on a particular day in a discussion board is more than three standard deviations higher than the previous mean. Figure 9 illustrates the result of this analysis for a selection of discussion boards from the reddit dataset. We see that there are discussion boards with relatively low variance in the controversiality over time such as *anime* and *askreddit*. However, there are also boards with spikes in controversiality such as *cringeepics* and *chicagobulls* which get marked by our anomaly detection approach. Manual investigation of these spikes revealed that they are strongly correlated to the number of deleted users on the particular day, which indicates that they indeed result from trolling behavior.

6. LEARNINGS

We report on learnings on different levels that we gained from users of our data validation system. On an *organisational level*, there are many benefits in using a common data quality library. Such a common library helps establish a shared vocabulary across teams for discussing data quality and also establishes best practices on how to measure data quality, leading to a common way to monitor the metrics of datasets. It is also a big advantage if producers as well as consumers of datasets leverage the same system for verifying data quality, as they can re-use checks and constraints from each other, e.g., the producer can choose to adapt checks from downstream consumers earlier in the data processing pipeline.

On a *technical level*, users highlighted the fact that our data quality library runs on Spark, which they experienced as a fast, scalable way to do data processing, partly due to the optimizations that our platform is applying. Our sys-

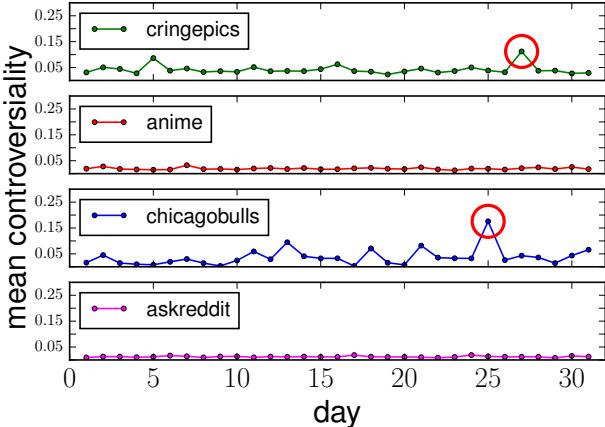


Figure 9: Anomalies detected in the time series of the Mean(controversiality) metric for different boards in the reddit dataset, which indicate trolling behavior potentially decreasing data quality.

tem helped reduce manual and ad-hoc analysis on their data, e.g., sampling and eyeballing the results to identify possible problems such as incomplete fields, outliers and derivations from the expected number of rows. Instead, such checks can now be run in an automated way as a part of ingestion pipelines. Additionally, data producers can leverage our system to halt their data publishing pipelines when they encounter cases of data anomalies. By that, they can ensure that downstream data processing, which often includes training ML models, is only working with vetted data.

7. RELATED WORK

Data cleaning has been an active research area for decades, see recent surveys [1, 10, 38] for an overview.

Declarative data quality verification The idea to allow declarative definitions of data quality standards is well-established. Ilyas et al. provide an overview on standard consistency definitions [23]. In fact, every DBMS supports standard *integrity constraints* such as key constraints or nullable fields. A relevant extension are *denial constraints*, which is a first order logic formalism that allows the coverage of more business rules across two tuples [11]. A popular paradigm for defining dependencies between columns are *functional dependencies* and *conditional functional dependencies* [6]; there exist fast, approximate algorithms for discovering them [31]. In contrast to this line of work, our predictability metric relies on ML to learn relationships between columns and uses empirical tests on hold-out datasets for validation. We conjecture that, due to their inherent robustness to outliers, ML methods more suitable for our use case to automatically detect changes in data quality on many large datasets. Galhardas et al. propose a declarative language *AJAX* for data cleaning as an extension of SQL as well as an execution model for executing data cleaning programs [14]. We similarly optimize the validation of our declarative data quality constraints to minimize computational effort. We combine a larger set of constraints into a unified framework, but we do not support the automatic execution of data repair methods.

ML for data cleaning Multiple researchers have suggested to use ML for cleaning data. While traditional methods can be used to generate candidates for fixing incorrect data (e.g., violations of functional dependencies), active learning methods can be used to select and prioritize human effort [49]. *ActiveClean* similarly uses active learning for prioritization, but at the same time it learns and updates a convex loss model [24]. *HoloClean* generates a probabilistic model over a dataset that combines integrity constraints and external data sources to generate data repair suggestions [35]. *Boost-Clean* automatically selects an ensemble of error detection and repair combinations using statistical boosting [25].

Data validation in ML applications The challenges involved in building complex machine learning applications in the real-world have recently been highlighted by many researchers, e.g., with respect to managing the resulting models [26], software engineering [42, 8], pipeline abstractions [2, 43, 46], and learnings from real-world systems [34, 7]. A new focus is being put on data management questions with respect to large-scale machine learning. Examples include managing the metadata of artifacts produced during machine learning workloads, including schemas and summary statistics of the datasets used for training and testing [47, 48, 40, 29, 28, 20, 33, 41], the discovery and organization of enterprise datasets [16], and machine learning specific sanity checks [45]. As a consequence, modern machine learning platforms begin to have explicit data validation components [7, 5, 9].

8. CONCLUSION

We presented a system for automating data quality verification tasks, which scales to large datasets and meets the requirements of production use cases. The system provides a declarative API to its users, which combines common quality constraints with custom validation code, and thereby enables ‘unit tests’ for data. We discussed how to efficiently execute the constraint validation by translating checks to scalable metrics computations, and elaborated on reformulations of the metrics to enable incremental computations on growing datasets. Additionally, we provided examples for the use of machine learning techniques in data quality verification, e.g. for enhancing constraint suggestions, for estimating the predictability of a column and for detecting anomalies in historic data quality timeseries.

In the future, we want to extend our machine learning-based constraint suggestion by leveraging more metadata as well as historical data about constraints defined with our API. Moreover, we will investigate the benefits of fitting well-known distributions to numeric columns to be able to understand this data in more detail and suggest more fine-grained constraints. Another direction is to provide users with more comprehensive error messages for failing checks and allow them easy access to records that made a particular constraint fail. Furthermore, we will apply seasonal ARIMA [22] and neural network-based time series forecasting [13] in order to enhance our anomaly detection functionality to also be able to handle seasonal and intermittent time series. Finally, we will explore validating data quality in streaming scenarios, which should be a natural extension of our discussed incremental use case.

9. REFERENCES

- [1] Z. Abedjan, L. Golab, and F. Naumann. Profiling relational data: a survey. *VLDB Journal*, 24(4):557–581, 2015.
- [2] P. Andrews, A. Kalro, H. Mehanna, and A. Sidorov. Productionizing Machine Learning Pipelines at Scale. *Machine Learning Systems workshop at ICML*, 2016.
- [3] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, et al. Spark sql: Relational data processing in spark. *SIGMOD*, 1383–1394, 2015.
- [4] C. Batini, C. Cappiello, C. Francalanci, and A. Maurino. Methodologies for data quality assessment and improvement. *ACM Computing Surveys*, 41(3):16, 2009.
- [5] D. Baylor, E. Breck, H.-T. Cheng, N. Fiedel, C. Y. Foo, Z. Haque, S. Haykal, M. Ispir, V. Jain, L. Koc, et al. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. *KDD*, 1387–1395, 2017.
- [6] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. *ICDE*, 746–755, 2007.
- [7] J.-H. Böse, V. Flunkert, J. Gasthaus, T. Januschowski, D. Lange, D. Salinas, S. Schelter, M. Seeger, and Y. Wang. Probabilistic demand forecasting at scale. *PVLDB*, 10(12):1694–1705, 2017.
- [8] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley. The ml test score: A rubric for ml production readiness and technical debt reduction. *Big Data*, 1123–1132, 2017.
- [9] E. Breck, N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich. Data Infrastructure for Machine Learning. *SysML*, 2018.
- [10] X. Chu, I. F. Ilyas, S. Krishnan, and J. Wang. Data cleaning: Overview and emerging challenges. *SIGMOD*, 2201–2206, 2016.
- [11] X. Chu, I. F. Ilyas, and P. Papotti. Discovering denial constraints. *PVLDB*, 6(13):1498–1509, 2013.
- [12] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining database structure; or, how to build a data quality browser. *SIGMOD*, 240–251, 2002.
- [13] V. Flunkert, D. Salinas, and J. Gasthaus. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *CorR*, abs/1704.04110, 2017.
- [14] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C.-A. Saita. Declarative data cleaning: Language, model, and algorithms. *VLDB*, 371–380, 2001.
- [15] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. *SIGMOD Record*, 30:58–66, 2001.
- [16] A. Halevy, F. Korn, N. F. Noy, C. Olston, N. Polyzotis, S. Roy, and S. E. Whang. GOODS: Organizing Google’s Datasets. *SIGMOD*, 795–806, 2016.
- [17] A. Y. Halevy, P. Norvig, and F. Pereira. The unreasonable effectiveness of data. *Intelligent Systems*, 24(2):8–12, 2009.
- [18] H. Harmouch and F. Naumann. Cardinality estimation: An experimental survey. *PVLDB*, 11(4):499–512, 2017.
- [19] J. M. Hellerstein. Quantitative data cleaning for large databases. *United Nations Economic Commission for Europe (UNECE)*, 2008.
- [20] J. M. Hellerstein, V. Sreekanti, J. E. Gonzalez, J. Dalton, A. Dey, S. Nag, K. Ramachandran, S. Arora, A. Bhattacharyya, S. Das, et al. Ground: A data context service. *CIDR*, 2017.
- [21] S. Heule, M. Nunkesser, and A. Hall. Hyperloglog in practice: algorithmic engineering of a state of the art cardinality estimation algorithm. *EDBT*, 683–692, 2013.
- [22] R. J. Hyndman and G. Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2014.
- [23] I. F. Ilyas and X. Chu. Trends in cleaning relational data: Consistency and deduplication. *Foundations and Trends in Databases*, 5(4), 281–393, 2015.
- [24] S. Krishnan, M. J. Franklin, K. Goldberg, J. Wang, and E. Wu. Activeclean: An interactive data cleaning framework for modern machine learning. *SIGMOD*, 2117–2120, 2016.
- [25] S. Krishnan, M. J. Franklin, K. Goldberg, and E. Wu. Boostclean: Automated error detection and repair for machine learning. *CorR*, abs/1711.01299, 2017.
- [26] A. Kumar, R. McCann, J. Naughton, and J. M. Patel. Model Selection Management Systems: The Next Frontier of Advanced Analytics. *SIGMOD Record*, 44(4):17–22, 2016.
- [27] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, et al. Mllib: Machine learning in Apache Spark. *JMLR*, 17(1):1235–1241, 2016.
- [28] H. Miao, A. Li, L. S. Davis, and A. Deshpande. On model discovery for hosted data science projects. *Workshop on Data Management for End-to-End Machine Learning at SIGMOD*, 6, 2017.
- [29] H. Miao, A. Li, L. S. Davis, and A. Deshpande. Towards unified data and lifecycle management for deep learning. *ICDE*, 571–582, 2017.
- [30] F. Naumann. *Quality-driven Query Answering for Integrated Information Systems*. Springer, 2002.
- [31] T. Papenbrock and F. Naumann. A hybrid approach to functional dependency discovery. *SIGMOD*, 821–833, 2016.
- [32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *JMLR*, 12:2825–2830, 2011.
- [33] J. F. Pimentel, L. Murta, V. Braganholo, and J. Freire. noworkflow: a tool for collecting, analyzing, and managing provenance from python scripts. *PVLDB*, 10(12):1841–1844, 2017.
- [34] N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich. Data management challenges in production machine learning. *SIGMOD*, 1723–1726, 2017.
- [35] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré. Holoclean: Holistic data repairs with probabilistic inference. *PVLDB*, 10(11):1190–1201, 2017.

- [36] J. D. Rennie, L. Shih, J. Teevan, and D. R. Karger. Tackling the poor assumptions of naive bayes text classifiers. *ICML*, 616–623, 2003.
- [37] T. Rukat, D. Lange, and C. Archambeau. An interpretable latent variable model for attribute applicability in the amazon catalogue. *Interpretable ML Symposium at NIPS*, 2017.
- [38] S. Sadiq, J. Freire, R. J. Miller, T. Dasu, I. F. Ilyas, F. Naumann, D. Srivastava, X. L. Dong, S. Link, and X. Zhou. Data quality: the role of empiricism. *SIGMOD Record*, 46(4):35–43, 2018.
- [39] M. Scannapieco and T. Catarci. Data quality under a computer science perspective. *Archivi & Computer*, 2, 1–15, 2002.
- [40] S. Schelter, J.-H. Boese, J. Kirschnick, T. Klein, and S. Seufert. Automatically Tracking Metadata and Provenance of Machine Learning Experiments. *Machine Learning Systems workshop at NIPS*, 2017.
- [41] S. Schelter, J.-H. Boese, J. Kirschnick, T. Klein, and S. Seufert. Declarative Metadata Management: A Missing Piece in End-to-End Machine Learning. *SysML*, 2018.
- [42] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J. Crespo, and D. Dennison. Hidden Technical Debt in Machine Learning Systems. *NIPS*, 2503–2511, 2015.
- [43] E. R. Sparks, S. Venkataraman, T. Kaftan, M. J. Franklin, and B. Recht. KeystoneML: Optimizing Pipelines for Large-Scale Advanced Analytics. *ICDE*, 535–546, 2017.
- [44] C. Sun, A. Shrivastava, S. Singh, and A. Gupta. Revisiting unreasonable effectiveness of data in deep learning era. *ICCV*, 843–852, 2017.
- [45] M. Terry, D. Sculley, and N. Hynes. The Data Linter: Lightweight, Automated Sanity Checking for ML Data Sets. *Machine Learning Systems Workshop at NIPS*, 2017.
- [46] T. van der Weide, D. Papadopoulos, O. Smirnov, M. Zielinski, and T. van Kasteren. Versioning for end-to-end machine learning pipelines. Workshop on Data Management for End-to-End Machine Learning at SIGMOD, 2, 2017.
- [47] J. Vanschoren, J. N. Van Rijn, B. Bischl, and L. Torgo. OpenML: networked science in machine learning. *KDD*, 49–60, 2014.
- [48] M. Vartak, H. Subramanyam, W.-E. Lee, S. Viswanathan, S. Husnoo, S. Madden, and M. Zaharia. ModelDB: A System for Machine Learning Model Management. *Workshop on Human-In-the-Loop Data Analytics at SIGMOD*, 14, 2016.
- [49] M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas. Guided data repair. *PVLDB*, 4(5):279–289, 2011.
- [50] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. *HotCloud*, 95, 2010.

DATA VALIDATION FOR MACHINE LEARNING

Eric Breck¹ Neoklis Polyzotis¹ Sudip Roy¹ Steven Euijong Whang² Martin Zinkevich¹

ABSTRACT

Machine learning is a powerful tool for gleaning knowledge from massive amounts of data. While a great deal of machine learning research has focused on improving the accuracy and efficiency of training and inference algorithms, there is less attention in the equally important problem of monitoring the quality of data fed to machine learning. The importance of this problem is hard to dispute: errors in the input data can nullify any benefits on speed and accuracy for training and inference. This argument points to a data-centric approach to machine learning that treats training and serving data as an important production asset, on par with the algorithm and infrastructure used for learning.

In this paper, we tackle this problem and present a data validation system that is designed to detect anomalies specifically in data fed into machine learning pipelines. This system is deployed in production as an integral part of TFX(Baylor et al., 2017) – an end-to-end machine learning platform at Google. It is used by hundreds of product teams use it to continuously monitor and validate several petabytes of production data per day. We faced several challenges in developing our system, most notably around the ability of ML pipelines to soldier on in the face of unexpected patterns, schema-free data, or training/serving skew. We discuss these challenges, the techniques we used to address them, and the various design choices that we made in implementing the system. Finally, we present evidence from the system’s deployment in production that illustrate the tangible benefits of data validation in the context of ML: early detection of errors, model-quality wins from using better data, savings in engineering hours to debug problems, and a shift towards data-centric workflows in model development.

1 INTRODUCTION

Machine Learning (ML) is widely used to glean knowledge from massive amounts of data. The applications are ever-increasing and range from machine perception and text understanding to health care, genomics, and self-driving cars. Given the critical nature of some of these applications, and the role that ML plays in their operation, we are also observing the emergence of *ML platforms* (Baylor et al., 2017; Chandra, 2014) that enable engineering teams to reliably deploy ML pipelines in production.

In this paper we focus on the problem of validating the input data fed to ML pipelines. The importance of this problem is hard to overstate, especially for production pipelines. Irrespective of the ML algorithms used, data errors can adversely affect the quality of the generated model. Furthermore, it is often the case that the predictions from the generated models are logged and used to generate more data for training. Such feedback loops have the potential to am-

plify even “small” data errors and lead to gradual regression of model performance over a period of time. Therefore, it is imperative to catch data errors early, before they propagate through these complex loops and taint more of the pipeline’s state. The importance of error-free data also applies to the task of model understanding, since any attempt to debug and understand the output of the model must be grounded on the assumption that the data is adequately clean. All these observations point to the fact that we need to elevate data to a first-class citizen in ML pipelines, on par with algorithms and infrastructure, with corresponding tooling to continuously monitor and validate data throughout the various stages of the pipeline.

Data validation is neither a new problem nor unique to ML, and so we borrow solutions from related fields (e.g., database systems). However, we argue that the problem acquires unique challenges in the context of ML and hence we need to rethink existing solutions. We discuss these challenges through an example that reflects an actual data-related production outage in Google.

Example 1.1 Consider an ML pipeline that trains on new training data arriving in batches every day, and pushes a fresh model trained on this data to the serving infrastructure. The queries to the model servers (the serving data)

¹Google Research ²KAIST. Work done while at Google Research. Correspondence to: Neoklis Polyzotis <npolyzotis@google.com>.

are logged and joined with labels to create the next day’s training data. This setup ensures that the model is continuously updated and adapts to any changes in the data characteristics on a daily basis.

Now, let us assume that an engineer performs a (seemingly) innocuous code refactoring in the serving stack, which, however, introduces a bug that pins the value of a specific int feature to -1 for some slice of the serving data (e.g., imagine that the feature’s value is generated by doing a RPC into a backend system and the bug causes the RPC to fail, thus returning an error value). The ML model, being robust to data changes, continues to generate predictions, albeit at a lower level of accuracy for this slice of data.

Since the serving data eventually becomes training data, this means that the next version of the model gets trained with the problematic data. Note that the data looks perfectly fine for the training code, since -1 is an acceptable value for the int feature. If the feature is important for accuracy then the model will continue to under-perform for the same slice of data. Moreover, the error will persist in the serving data (and thus in the next batch of training data) until it is discovered and fixed.

The example illustrates a common setup where the generation (and ownership!) of the data is decoupled from the ML pipeline. This decoupling is often necessary as it allows product teams to experiment and innovate by joining data sources maintained and curated by other teams. However, this multiplicity of data sources (and corresponding code paths populating the sources) can lead to multiple failure modes for different slices of the data. A lack of visibility by the ML pipeline into this data generation logic except through side effects (e.g., the fact that -1 became more common on a slice of the data) makes detecting such slice-specific problems significantly harder. Furthermore, the pipeline often receives the data in a raw-value format (e.g., `tensorflow.Example` or CSV) that strips out any semantic information that can help identify errors. Going back to our example, -1 is a valid value for the int feature and does not carry with it any semantics related to the backend errors. Overall, there is little a-priori information that the pipeline can leverage to reason about data errors.

The example also illustrates a very common source of data errors in production: bugs in code. This has several important implications for data validation. First, data errors are likely to exhibit some “structure” that reflects the execution of the faulty code (e.g., all training examples in the slice get the value of -1). Second, these errors tend to be different than the type of errors commonly considered in the data-cleaning literature (e.g., entity resolution). Finally, fixing an error requires changes to code which is immensely hard to automate reliably. Even if it were possible to automatically “patch” the data to correct an error (e.g., using a technique

such as Holocleans (Rekatsinas et al., 2017)), this would have to happen consistently for both training and serving data, with the additional challenge that the latter is a stream with stringent latency requirements. Instead, the common approach in practice is for the on-call engineer to investigate the problem, isolate the bug in the code, and submit a fix for both the training and serving side. In turn, this means that the data-validation system must generate reliable alerts with high precision, and provide enough context so that the on-call engineer can quickly identify the root cause of the problem. Our experience shows that on-calls tend to ignore alerts that are either spammy or not actionable, which can cause important errors to go unnoticed.

A final observation on Example 1.1 is that errors can happen and propagate at different stages of the pipeline. Catching data errors early is thus important, as it helps debug the root cause and also rollback the pipeline to a working state. Moreover, it is important to rely on mechanisms specific to data validation rather than on detection of second-order effects. Concretely, suppose that we relied on model-quality validation as a failsafe for data errors. The resilience of ML algorithms to noisy data means that errors may result in a small drop in model quality, one that can be easily missed if the data errors affect the model only on specific slices of the data but the aggregate model metrics still look okay.

Our Contributions. In this paper we present a data-validation system whose design is driven by the aforementioned challenges. Our system is deployed in production as an integral part of TFX. Hundreds of product teams use our system to validate trillions of training and serving examples per day, amounting to several petabytes of data per day.

The data-validation mechanisms that we develop are based on “battle-tested” principles from data management systems, but tailored to the context of ML. For instance, a fundamental piece of our solution is the familiar concept of a data schema, which codifies the expectations for correct data. However, as mentioned above, existing solutions do not work out of the box and need to adapt to the context of ML. For instance, the schema encodes data properties that are unique to ML and are thus absent from typical database schemas. Moreover, the need to surface high-precision, actionable alerts to a human operator has influenced the type of properties that we can check. As an example, we found that statistical tests for detecting changes in the data distribution, such as the chi-squared test, are too sensitive and also uninformative for the typical scale of data in ML pipelines, which led us to seek alternative methods to quantify changes between data distributions. Another difference is that in database systems the schema comes first and provides a mechanism to verify both updates and queries against the data. This assumption breaks in the context of ML pipelines where data generation is decoupled from the pipeline. In

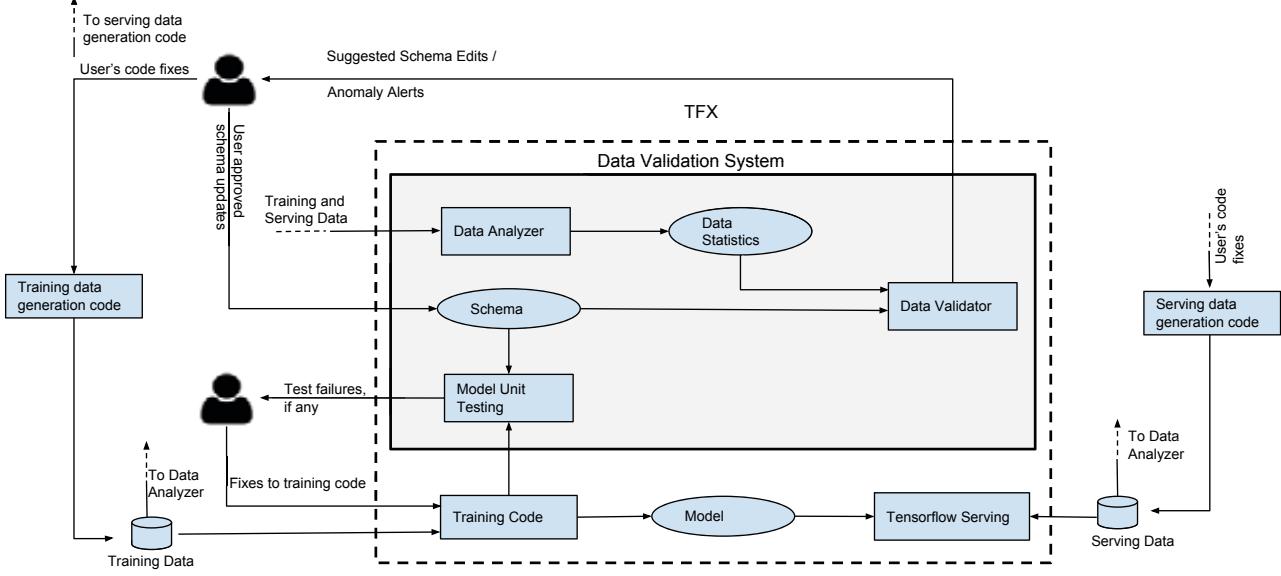


Figure 1: An overview of our data validation system and its integration with TFX.

essence, in our setup the data comes first. This necessitates new workflows where the schema can be inferred from and co-evolve with the data. Moreover, this co-evolution needs to be user friendly so that we can allow human operators to seamlessly encode their domain knowledge about the data as part of the schema.

Another novel aspect of our work is that we use the schema to perform unit tests for the training algorithm. These tests check that there are no obvious errors in the code of the training algorithm, but most importantly they help our system uncover any discrepancies between the codified expectations over the data and the assumptions made by the algorithm. Any discrepancies mean that either the schema needs to change (to codify the new expectations) or the training code needs to change to be compliant with the actual shape of the data. Overall, this implies another type of co-evolution between the schema and the training algorithm.

As mentioned earlier, our system has been deployed in production at Google and so some parts of its implementation have been influenced by the specific infrastructure that we use. Still, we believe that our design choices, techniques, and lessons learned generalize to other setups and are thus of interest to both researchers and practitioners. We have also open-sourced the libraries¹ that implement the core techniques that we describe in the paper so that they can be integrated in other platforms.

¹Github repository for TensorFlow Data Validation: <https://github.com/tensorflow/data-validation>

2 SYSTEM OVERVIEW

Figure 1 shows a schematic overview of our data-validation system and how it integrates with an end-to-end machine learning platform. At a high level, the platform instantiates a pipeline that ingests training data, passes it to data validation (our system), then pipes it to a training algorithm that generates a model, and finally pushes the latter to a serving infrastructure for inference. These pipelines typically work in a continuous fashion: a new batch of data arrives periodically, which triggers a new run of the pipeline. In other words, each batch of input data will trigger a new run of the data-validation logic and hence potentially a new set of data anomalies. Note that our notion of a batch is different than the mini-batches that constitute chunks of data used to compute and update model parameters during training. The batches of data ingested into the pipeline correspond to larger intervals of time (say a day). While our system supports validation of data on a sample of data ingested into the pipeline, this option is disabled by default since our current response times for single batch is acceptable for most users and within the expectations of end-to-end ML. Furthermore, by validating over the entire batch we ensure that anomalies that are infrequent or manifest in small but important slices of data are not silently ignored.

The data validation system consists of three main components – a *Data Analyzer* that computes predefined set of data statistics sufficient for data validation, a *Data Validator* that checks for properties of data as specified through a

Schema (defined in Section 3), and a *Model Unit Tester* that checks for errors in the training code using synthetic data generated through the schema. Overall, our system supports the following types of data validation:

- **Single-batch validation** answers the question: are there any anomalies in a single batch of data? The goal here is to alert the on-call about the error and kick-start the investigation.
- **Inter-batch validation** answers the question: are there any significant changes between the training and serving data, or between successive batches of the training data? This tries to capture the class of errors that occur between software stacks (e.g., training data generation follows different logic than serving data generation), or to capture bugs in the rollout of new code (e.g., a new version of the training-data generation code results in different semantics for a feature, which requires old batches to be backfilled).
- **Model testing** answers the question: are there any assumptions in the training code that are not reflected in the data (e.g. are we taking the logarithm of a feature that turns out to be a negative number or a string?).

The following sections discuss the details of how our system performs these data validation checks and how our design choices address the challenges discussed in Section 1. Now, we acknowledge that these checks are not exhaustive, but our experience shows that they cover the vast majority of errors we have observed in production and so they provide a meaningful scope for the problem.

3 SINGLE-BATCH VALIDATION

The first question we answer is: are there data errors within each new batch that is ingested by the pipeline?

We expect the data characteristics to remain stable within each batch, as the latter corresponds to a single run of the data-generation code. We also expect some characteristics to remain stable across several batches that are close in time, since it is uncommon to have frequent *drastic* changes to the data-generation code. For these reasons, we consider any deviation within a batch from the expected data characteristics, given expert domain knowledge, as an *anomaly*.

In order to codify these expected data characteristics, our system generalizes the traditional notion of a *schema* from database systems. The schema acts as a logical model of the data and attempts to capture some of the semantics that are lost when the data are transformed to the format accepted by training algorithms, which is typically key-value lists. To illustrate this, consider a training example with a key-value (“age”, 150). If this feature corresponds to the age of a

person in years then clearly there is an error. If, however, the feature corresponds to the age of a document in days then the value can be perfectly valid. The schema can codify our expectations for “age” and provide a reliable mechanism to check for errors.

```
message Schema {
    // Features described in this schema.
    repeated Feature feature;

    // String domains referenced in the features.
    repeated StringDomain string_domain;
}

message Feature {
    // The name of the feature.
    string name;

    // Type of the feature's values
    FeatureType type;

    // Constraints on the number of examples that have this
    // feature populated.
    FeaturePresence presence;

    // Minimum and maximum number of values.
    ValueCount value_count;

    // Domain for the values of the feature.
    oneof domain_info {
        // Reference to a domain defined at the schema
        // level.
        string domain;

        // Inline definitions of domains.
        IntDomain int_domain;
        FloatDomain float_domain;
        StringDomain string_domain;
        BoolDomain bool_domain;
    }

    LifecycleStage lifecycle_stage;
}
```

Figure 2: Simplified schema as a protocol buffer. Note that tag numbers are omitted to simplify exposition. For further explanation of constraints, see Appendix A.

Figure 2 shows the schema formalism used by our data validation system, defined as a protocol buffer message([pro, 2017](#)). (We use the protocol-buffer representation as it corresponds to our implementation and is also easy to follow.) The schema follows a logical data model where each training or serving example is a collection of features, with each feature having several constraints attached to it. This flat model has an obvious mapping to the flat data formats used by popular ML frameworks, e.g., tensorflow.Example or CSV. We have also extended the schema to cover structured examples (e.g., JSON objects or protocol buffers) but we do not discuss this capability in this paper.

The constraints associated with each feature cover some basic properties (e.g., type, domain, valency) but also constraints that are relevant to ML and that also reflect code-centric error patterns that we want to capture through data validation (see also our discussion in Section 1). For instance, the presence constraint covers bugs that cause the data-generation code to silently drop features from some examples (e.g., failed RPCs that cause the code to skip a feature). As another case, the domain constraints can cover bugs that change the representation of values for the same

feature (e.g., a code-change that lower-cases the string representation of country codes for some examples). We defer a more detailed discussion of our schema formalism in Appendix A. However, we note that we do not make any claims on completeness – there are reasonable data properties that our schema cannot encode. Still, our experience so far has shown that the current schema is powerful enough to capture the vast majority of use cases in production pipelines.

Schema Validation The *Data Validator* component of our system validates each batch of data by comparing it against the schema. Any disagreement is flagged as an anomaly and triggers an alert to the on-call for further investigation. For now, we assume that the pipeline includes a user-curated schema that codifies all the constraints of interest. We discuss how to arrive at this state later.

In view of the design goals of data validation discussed in Section 1, the Data Validator component:

- attempts to detect issues as early in the pipeline as possible to avoid training on bad data. In order to ensure it can do so scalably and efficiently, we rely on the per-batch data statistics computed by a preceding *Data Analyzer* module. Our strategy of validating using these pre-computed statistics allows the validation itself to be fairly lightweight. This enables revalidation of the data using an updated schema near instantaneously.
- be easily interpretable and narrowly focused on the exact anomaly. Table 1 shows different categories of anomalies that our Data Validator reports to the user. Each anomaly corresponds to a violation of some property specified in the schema and has an associated description template that is populated with concrete values from the detected anomaly before being presented to the user.
- include suggested updates to the schema to “eliminate” anomalies that correspond to the natural evolution of the data. For instance, as shown in Figure 4, the domain of

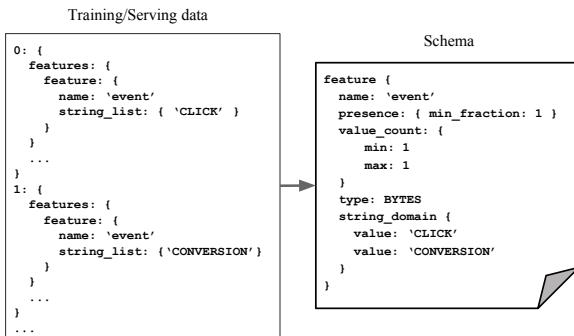


Figure 3: An example schema and corresponding data in the `tf.train.Example` (`tfe`) format.

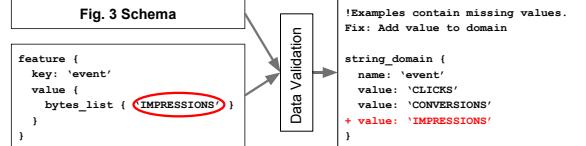


Figure 4: Schema-driven validation

“event” seems to have acquired a new IMPRESSIONS value, and so our validator generates a suggestion to extend the feature’s domain with the same value (this is shown with the red text and “+” sign in Figure 4).

- avoids false positives by focusing on the narrow set of constraints that can be expressed in our schema and by requiring that the schema is manually curated, thus ensuring that the user actually cares about the encoded constraints.

Schema Life Cycle As mentioned earlier, our assumption is that the pipeline owners are also responsible to curate the schema. However, many machine learning pipelines use thousands of features, and so constructing a schema manually for such pipelines can be quite tedious. Furthermore, the domain knowledge of the features may be distributed across a number of engineers within the product team or even outside of the team. In such cases, the upfront effort in schema construction can discourage engineers from setting up data validation until they run into a serious data error.

To overcome this adoption hurdle, the Data Validator component synthesizes a basic version of the schema based on all available batches of data in the pipeline. This auto-generated schema attempts to capture salient properties of the data without overfitting to a particular batch of data. Avoiding overfitting is important: an overfitted schema is more likely to cause spurious alerts when validating a new batch of data, which in turn increases the cognitive overhead for the on-call engineers, reduces their trust in the system, and may even lead them to switch off data validation altogether. We currently rely on a set of reasonable heuristics to perform this initial schema inference. A more formal solution, perhaps with guarantees or controls on the amount of overfitting, is an interesting direction for future work.

Once the initial schema is in place, the Data Validator will recommend updates to the schema as new data is ingested and analyzed. To help users manage the schema easily, our system also includes a user interface and tools that aid users by directing their attention to important suggestions and providing a click-button interface to apply the suggested changes. The user can then accept these suggested updates or manually edit the schema using her judgement. We expect owners of pipelines to treat the schema as a production asset at par with source code and adopt best practices for reviewing, versioning, and maintaining the schema. For instance, in our pipelines the schema is stored in the version-

control system for source code.

4 INTER-BATCH VALIDATION

There are certain anomalies that only manifest when two or more batches of data are considered together, e.g., drift in the distribution of feature values across multiple batches of data. In this section, we first cover the different types of such anomalies, discuss the reasons why they occur, and finally present some common techniques that can be used to detect them.

Training-Serving Skew One of the issues that frequently occurs in production pipelines is skew between training and serving data. Several factors contribute to this issue but the most common is different code paths used for generation of training and serving data. These different code paths are required due to widely different latency and throughput characteristics of offline training data generation versus online serving data generation.

Based on our experience, we can broadly categorize training-serving skew into three main categories.

Feature skew occurs when a particular feature for an example assumes different values in training versus at serving time. This can happen, for instance, when a developer adds or removes a feature from the training-data code path but inadvertently forgets to do the same to the serving path. A more interesting mechanism through which feature skew occurs is termed *time travel*. This happens when the feature value is determined by querying a non-static source of data. For instance, consider a scenario where each example in our data corresponds to an online ad impression. One of the features for the impression is the number of clicks, obtained by querying a database. Now, if the training data is generated by querying the same database then it is likely that the click count for each impression will appear higher compared to the serving data, since it includes all the clicks that happened between when the data was served and when the training data was generated. This skew would bias the resulting model against a different distribution of click rates compared to what is observed at serving time, which is likely to affect model quality.

Distribution skew occurs when the distribution of feature values over a batch of training data is different from that seen at serving time. To understand why this happens consider the following scenario. Let us assume that we have a setup where a sample of today’s serving data is used as the training data for next day’s model. The sampling is needed since the volume of data seen at serving time is prohibitively large to train over. Any flaw in the sampling scheme can result in training data distributions that look significantly different from serving data distributions.

Scoring/Serving Skew occurs when only a subset of the scored examples are actually served. To illustrate this scenario, consider a list of ten recommended videos shown to the user out of hundred that are scored by the model. Subsequently if the user clicks on one of the ten videos, then we can treat that as a positive example and the other nine as negative examples for next day’s training. However, the ninety videos that were never served may not have associated labels and therefore may never appear in the training data. This establishes an implicit feedback loop which further increases the chances of misprediction for lower ranked items, and consequently less frequent appearance in training data. Detecting such types of skew is harder than the other two types.

In addition to validating individual batches of training data, the *Data Validator* also monitors for skew between training and serving data, continuously. Specifically, our serving infrastructure is configured to log a sample of the serving data which is imported back into the training pipeline. The Data Validator component continuously compares batches of incoming training and serving data to detect different types of skew. To detect feature skew, the Data Validator component does a key-join between corresponding batches of training and serving data followed by a feature wise comparison. Any detected skew is summarized and presented to the user using the ML platform’s standard alerting infrastructure. To detect distribution skew, we rely on measures that quantify the distance between distributions, which we discuss below. In addition to skew detection, each serving data batch is validated against the schema to detect the anomalies listed in Section 3. We note that in all cases, the parameters for skew detection are encoded as additional constraints in the schema.

Quantifying Distribution Distance As mentioned above, our distribution-skew detection relies on metrics that can quantify the distance between the training and serving distributions. It is important to note that, in practice, we expect these distributions to be different (and hence their distance to be positive), since the distribution of examples on each day is in fact different than the last. So the key question is whether the distance is high enough to warrant an alert.

A first-cut solution here might be to use typical distance metrics such as KL divergence or cosine similarity and fire an alert only if the metric crosses a threshold. The problem with this solution is that product teams have a hard time understanding the natural meaning of the metric and thus tuning the threshold to avoid false positives. Another approach might be to rely on statistical goodness-of-fit tests, where the alert can be tuned based on common confidence levels. For instance, one might apply a chi-square test and alert if the null hypothesis is rejected at the 1% confidence level. The problem with this general approach, however, is that the variance of test statistics typically has an inverse

square relationship to the total number of examples, which makes these tests sensitive to even minute differences in the two distributions.

We illustrate the last point with a simple experiment. We begin with a control sample of 100 million points from a $N(0,1)$ distribution and then randomly replace 10 thousand points by sampling from a $N(0,2)$ distribution. This replacement introduces a minute amount of noise (0.01%) in the original data, for which ML is expected to be resilient. In other words, a human operator would consider the two distributions to be the “same”. We then perform a chi-square test between the two distributions to test their fit. Figure 5 shows the p-value of this test over 10 trials of this experiment, along with the threshold of 1% for the confidence level. Note that any p-value below the threshold implies that the test rejects the null hypothesis (that the two distributions are the same) and would therefore result in a distribution-skew alert. As shown, this method would needlessly fire an alert 7 out of 10 times and would most likely be considered a flaky detection method. We have repeated the same experiment with actual error-free data from one of our product teams and obtained qualitatively the same results.

To overcome these limitations we focus on distribution-distance metrics which are resilient to differences we observe in practice and whose sensitivity can be easily tuned by product teams. In particular, we use $d_\infty(p, q) = \max_{i \in S} |p_i - q_i|$ as the metric to compare two distributions p and q with probabilities p_i and q_i respectively for each value i (from some universe). Notice that this metric has several desirable properties. First, it has a simple natural interpretation of the largest change in probability for a value in the two distributions, which makes it easier for teams to set a threshold (e.g., “allow changes of only up to 1% for each value”). Moreover, an alert comes with a “culprit” value that can be the starting point of an investigation. For instance, if the highest change in frequency is observed in value ‘-1’ then this might be an issue with some backend failing and producing a default value. (This is an actual situation observed in one of our production pipelines.)

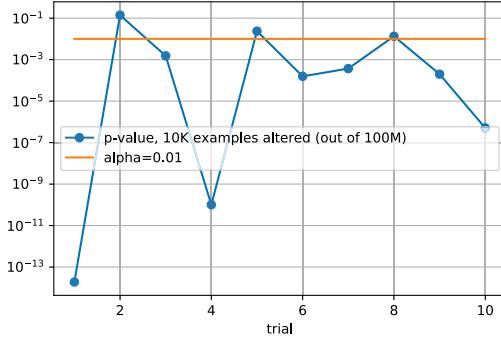


Figure 5: Sensitivity of chi-square test.

The next question is whether we can estimate the metric in a statistically sound manner based on observed frequencies. We develop a method to do that based on Dirichlet priors. For more details, see Appendix B.

5 MODEL UNIT TESTING

Up to this point, we focused on detecting mismatches between the expected and the actual state of the data. Our intent has been to uncover software errors in generating either training or serving data. Here, we shift gears and focus on a different class of software errors: mismatches between the expected data and the assumptions made in the training code. Specifically, recall that our platform (and similar platforms) allow the user to plug in their own training code to generate the model (see also Figure 1). This code is mostly a black box for the remaining parts of the platform, including the data-validation system, and can perform arbitrary computations over the data. As we explain below, these computations may make assumptions that do not agree with the data and cause serious errors that propagate through the ML infrastructure.

To illustrate this scenario, suppose that the training code applies a logarithm transform on a numeric feature, thus making the implicit assumption that the feature’s value is always positive. Now, let us assume that the schema does not encode this assumption (e.g., the feature’s domain includes non-positive values) and also that the specific instance of training data happens to carry positive values for this feature. As a result, two things happen: (a) data validation does not detect any errors, and (b) the generated model includes the same transform and hence the same assumption. Consider now what happens when the model is served and the serving data happens to have a non-positive value for the feature: the error is triggered, resulting in a (possibly crashing) failure inside the serving infrastructure.

The previous example illustrates the dangers of these hidden assumptions and the importance of catching them before they propagate through the served models. Note that the training code can make several other types of assumptions that are not reflected in the schema, e.g., that a feature is always present even though it is marked as optional, or that a feature has a dense shape even though it may take a variable number of values, to name a few that we have observed in production. Again, the danger of these assumptions is that they are not part of the schema (so they cannot be caught through schema validation) and they may be satisfied in the specific instance of training data (and so will not trigger during training).

We base our approach on fuzz testing (Miller et al., 1990), using the schema to guide the generation of synthetic inputs. Specifically, we generate synthetic training examples

that adhere to the schema constraints. For instance, if the schema in Figure 3 is used to generate data, each example will have an *event* feature, each of which would have one value, uniformly chosen at random to be either ‘CLICK’ or ‘CONVERSION’. Similarly, integral features would be random integers from the range specified in the schema, to name another case. The generation can be seeded so that it provides a deterministic output.

The generated data is then used to drive a few iterations of the training code. The goal is to trigger the hidden assumptions in the code that do not agree with the schema constraints. Returning to our earlier example with the logarithm transform, a synthetic input with non-positive feature values will cause an error and hence uncover the mismatch between the schema and the training code. At that point, the user can fix the training code, e.g., apply the transform on $\max(\text{value}, 1)$, or extend the schema to mark the feature as positive (so that data validation catches this error). Doing both provides for additional robustness, as it enables alerts if the data violates the stated assumptions and protects the generated model from crashes if the data is wrong.

This fuzz-testing strategy is obviously not fool-proof, as the random data can still pass through the training code without triggering errors. In practice, however, we found that fuzz-testing can trigger common errors in the training code even with a modest number of randomly-generated examples (e.g., in the 100s). In fact, it has worked so well that we have packaged this type of testing as a unit test over training algorithms, and included the test in the standard templates of our ML platform. Our users routinely execute these unit tests to validate changes to the training logic of their pipelines. To our knowledge, this application of unit testing in the context of ML and for the purpose of data validation is a novel aspect of our work.

6 EMPIRICAL VALIDATION

As mentioned earlier, our data-validation system has been deployed in production as part of the standard ML platform used in a large organization. The system currently analyzes several petabytes of training and serving data per day and has resulted in significant savings in engineer hours in two ways: by catching important data anomalies early (before they result in a bad model being pushed to production), and by helping teams diagnose model-quality problems caused by data errors. In what follows we provide empirical evidence to back this claim. We first report aggregate results from our production deployment and then discuss in some detail individual use cases.

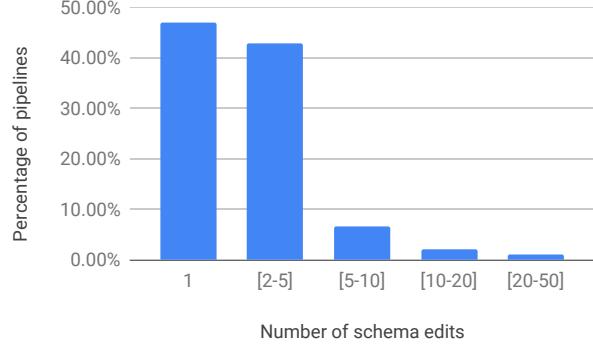


Figure 6: Number of manual schema changes by users over the analysis pipelines.

6.1 Results from Production Deployment

We present an analysis of our system in production, based on a sample of more than 700 ML pipelines that employ data validation and train models that are pushed to our production serving infrastructure.

We first consider the co-evolution of data and schema in the lifecycle of these pipelines, which is a core tenet of our approach. Recall that our users go through the following workflow: (i) when the pipeline is set up, they obtain an initial schema through our automatic inference, (ii) they review and manually edit the schema to compensate for any missing domain knowledge, (iii) they commit the updated schema to a version control system and start the training part of the pipeline. Subsequently, when a data alert fires that indicates a disagreement between the data and the schema, the user has the option to either fix the data generation process or update the schema, if the alert corresponds to a natural evolution of the data. Here we are interested in understanding the latter changes that reflect the co-evolution we mentioned.

Figure 6 shows a histogram of this number of schema changes across the >700 pipelines that we considered. The graph illustrates that the schema evolves in most of the examined pipelines, in line with our hypothesis of data-schema co-evolution.

The majority of cases have up to five schema revisions, with the distribution tapering off after that point. This evidence suggests that the input data has evolving properties but is not completely volatile, which is reasonable in practice. On the side, we also conclude that the engineers treat the schema as an important production asset and hence are willing to put in the effort to keep it up to date. In fact, anecdotal evidence from some teams suggest a mental shift towards a data-centric view of ML, where the schema is not solely used for data validation but also provides a way to document new features that are used in the pipeline and thus disseminate

Anomaly Category	Used	Fired	Fixed given Fired
New feature column (in data but not in schema)	100%	10%	65%
Out of domain values for categorical features	45%	6%	66%
Missing feature column (in schema but not in data)	97%	6%	53%
The fraction of examples containing a feature is too small	97%	3%	82%
Too small feature value vector for example	98%	2%	56%
Too large feature value vector for example	98%	<1%	28%
Data completely missing	100%	3%	65%
Incorrect data type for feature values	98%	<1%	100%
Non-boolean value for boolean feature type	14%	<1%	100%
Out of domain values for numeric features	67%	1%	77%

Table 1: Analysis of data anomalies over the most recent 30-day period for evaluation pipelines. First, we checked the schemas, to determine what fraction of pipelines could possibly fire a particular kind of alert (Used). Then, we looked at each day, and saw what kinds of anomalies Fired, and calculated what fraction of pipelines had an anomaly fire on any day. If there were two days with none of this type of anomaly firing on a pipeline afterward, then we considered the problem Fixed. This methodology can miss some fixes, if an anomaly is fixed but a new anomaly of the same type arrives the next day. It is also possible that an anomaly appears fixed but wasn’t if a pipeline stopped or example validation was turned off, but this is less likely.

information across the members of the team.

In our next analysis we examine in more detail how users interact with the schema. Specifically, we instrumented our system to monitor the types of data anomalies raised in production and the subsequent reactions of the users in terms of schema updates. Table 1 summarizes the results. As shown, the most common anomalies are new feature columns, unexpected string values, and missing feature columns. The first two are unsurprising: even in a healthy pipeline, there will be new values for fields such as “postal code”, and new feature columns are constantly being created by feature engineers. Missing features and missing data are more cause for concern. We can see from the chart that it is very rare that the physical type of a feature column is wrong (for example, someone manually added a feature column with the wrong type, causing this anomaly); nonetheless, by checking this we check agreement between the prescriptive nature of the schema and the actual data on disk. Even in cases where the anomalies almost never fire, this check is useful.

Table 1 also shows that product teams fix the majority of the detected anomalies. Now, some anomalies remain unfixed and we postulate that this happens for two reasons. First, in some cases a reasonable course of action is to let the anomalous batch slide out of the pipeline’s rolling window, e.g., in the case of a data-missing anomaly where regenerating the data is impossible. Second, as we mentioned repeatedly, we are dealing with a human operator in the loop who might miss or ignore the alert depending on their cognitive load. This re-emphasizes the observation that we need to be mindful of how and when we deliver the anomaly alerts to the on-call operators.

Finally, we turn our attention to the data-validation workflow through model unit testing. Again, this unit testing is part of the standard ML platform in our organization and so we were able to gather fleet-wide statistics on this functionality. Specifically, more than 70% of pipelines had at least one model unit test defined. Based on analysis of test logs over a period of one month, we determined that these tests were executed more than 80K times (including runs executed as part of continuous test framework). Of all of these executions 6% had failures indicating that either the training code had incorrect assumptions about the data or the schema itself was under specified.

6.2 Case Studies

In addition to the previous results, we present three case studies that illustrate the benefits of our system in production.

Missing features in Google Play recommender pipeline. The goal of the Google Play recommender system is to recommend relevant Android apps to the Play app users when they visit the homepage of the store, with an aim of driving discovery of apps that will be useful to the user. Using the Data Validation system of TFX, Google Play discovered a few features that were always missing from the logs, but always present in training. The results of an online A/B experiment showed that removing this skew improved the app install rate on the main landing page of the app store by 2%.

Data debugging leads to model wins. One of the product teams in our organization employs ML to generate video recommendations. The product team needed to migrate their existing training and serving infrastructure onto the new ML platform that also includes data validation. A prerequisite for the migration was of course to achieve parity in terms of model quality. After setting up the new infrastructure, our data-validation system started generating alerts about missing features over the serving data. During the ensuing investigation, the team discovered that a backend system was storing some features in a different format than expected which caused these features to be silently dropped in their

data-parsing code. In this case, the hard part was identifying the features which exhibited the problem (and this is precisely the information provided by our data-validation service)– the fix was easy and resulted in achieving performance parity. The total time to diagnose and fix the problem was two days, whereas similar issues have taken months to resolve.

Stand-alone data validation. Some product teams are setting up ML pipelines solely for the purpose of running our data-validation system, without doing any training or serving of models. These teams have existing infrastructure (predating the ML platform) to train and serve models, but they are lacking a systematic mechanism for data validation and, as a result, they are suffering from the data-related issues that we described earlier in the paper. To address this shortcoming in their infrastructure, these teams have set up pipelines that solely monitor the training and serving data and alert the on-call when an error is detected (while training and serving still happen using the existing infrastructure).

Feature-store migration. Two product teams have used our system to validate the migration of their features to new stores. In this setup, our system was used to apply schema validation and drift detection on datasets from the old and new storage systems, respectively. The detected data errors helped the teams debug the new storage system and consequently complete the migration successfully.

7 RELATED WORK

As compared to a similar system from Amazon(Schelter et al., 2018), our design choices and techniques are informed based on wide deployment of our system at Google. While Amazon’s system allows users to express arbitrary constraints, we opted to have a restrictive schema definition language that captures the data constraints for most of our users while permitting us to develop effective tools to help users manage their schema. Another differentiating factor of our work is the emphasis on co-evolution of the schema with data and the model with the user in the loop.

While model training (Abadi et al., 2016; ker; mxn) is a central topic, it is only one component of a machine learning platform and represents a small fraction of the code (Sculley et al., 2015). As a result, there is an increasing effort to build end-to-end machine learning systems (Chandra, 2014; Baylor et al., 2017; Sparks et al., 2017; Böse et al., 2017) that cover all aspects of machine learning including data management (Polyzotis et al., 2017), model management (Vartak, 2017; Fernandez et al., 2016; Schelter et al., 2017), and model serving (Olston et al., 2017; Crankshaw et al., 2015), to name a few. In addition, testing and monitoring are necessary to reduce technical debt and ensure product readiness of machine learning systems (Breck et al., 2017). In com-

parison, this paper specifically focuses on data validation for production machine learning. In addition we go into detail on a particular solution to the problem, compared to previous work that covered in broad strokes the issues related to data management and machine learning (Polyzotis et al., 2017).

A topic closely related to data validation is data cleaning (Rekatsinas et al., 2017; Stonebraker et al., 2013; Khayyat et al., 2015; Volkovs et al., 2014) where the state-of-art is to incorporate multiple signals including integrity constraints, external data, and quantitative statistics to repair errors. More recently, several cleaning tools target machine learning. BoostClean (Krishnan et al., 2017) selectively cleans out-of-range errors that negatively affect model accuracy. Data linter (Hynes et al., 2017) uses best practices to identify potential issues and efficiencies of machine learning data. In comparison, our system uses a data-driven schema utilizing previous data to generate actionable alerts to users. An interesting direction is to perform root cause analysis (Wang et al., 2015) that is actionable as well.

Schema generation is commonly done in Database systems (DiScala & Abadi, 2016) where the main focus is on query processing. As discussed in Section 1, our context is unique in that the schema is reverse-engineered and has to co-evolve with the data. Moreover, we introduce schema constraints unique to ML.

Skew detection is relevant to various statistical approaches including homogeneity tests (Pearson, 1992), analysis of variance (Fisher, 1921; 1992), and time series analysis (Basseville & Nikiforov, 1993; Ding et al., 2008; Brodersen et al., 2015). As discussed, our approach is to avoid statistical tests that lead to false positive alerts and instead rely on more interpretable metrics of distribution distance, coupled with sound approximation methods.

The traditional approach of model testing is to select a random set of examples from manually labeled datasets (Witten et al., 2011). More recently, adversarial deep learning techniques (Goodfellow et al., 2014) have been proposed to generate examples that fool deep neural networks. DeepXplore (Pei et al., 2017) is a whitebox tool for generating data where models make different predictions. There are also many tools for traditional software testing. In comparison, our model testing uses a schema to generate data and can thus work for any type of models. The process of iteratively adjusting the schema is similar in spirit to version spaces (Russell & Norvig, 2009) and has connections with teaching learners (Frazier et al., 1996).

REFERENCES

- Keras. <https://keras.io/>.
- Mxnet. <https://mxnet.incubator.apache.org/>.
- Tensorflow examples. https://www.tensorflow.org/programmers_guide/datasets.
- Protocol buffers. <https://developers.google.com/protocol-buffers/>, 2017.
- Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. Tensorflow: A system for large-scale machine learning. In *OSDI*, pp. 265–283, 2016. ISBN 978-1-931971-33-1.
- Basseville, M. and Nikiforov, I. V. *Detection of Abrupt Changes: Theory and Application*. Prentice-Hall, Inc., 1993. ISBN 0-13-126780-9.
- Baylor, D., Breck, E., Cheng, H.-T., Fiedel, N., Foo, C. Y., Haque, Z., Haykal, S., Ispir, M., Jain, V., Koc, L., Koo, C. Y., Lew, L., Mewald, C., Modi, A. N., Polyzotis, N., Ramesh, S., Roy, S., Whang, S. E., Wicke, M., Wilkiewicz, J., Zhang, X., and Zinkevich, M. TFX: A tensorflow-based production-scale machine learning platform. In *SIGKDD*, pp. 1387–1395, 2017. ISBN 978-1-4503-4887-4. doi: 10.1145/3097983.3098021. URL <http://doi.acm.org/10.1145/3097983.3098021>.
- Böse, J.-H., Flunkert, V., Gasthaus, J., Januschowski, T., Lange, D., Salinas, D., Schelter, S., Seeger, M., and Wang, Y. Probabilistic demand forecasting at scale. *PVLDB*, 10(12):1694–1705, August 2017. ISSN 2150-8097.
- Breck, E., Cai, S., Nielsen, E., Salib, M., and Sculley, D. The ml test score: A rubric for ml production readiness and technical debt reduction. In *IEEE Big Data*, 2017.
- Brodersen, K. H., Gallusser, F., Koehler, J., Remy, N., and Scott, S. L. Inferring causal impact using bayesian structural time-series models. *Annals of Applied Statistics*, 9: 247–274, 2015.
- Chandra, T. Sibyl: a system for large scale machine learning at Google. In *Dependable Systems and Networks (Keynote)*, Atlanta, GA, 2014. URL <http://www.youtube.com/watch?v=3SaZ5UAQrQM>.
- Crankshaw, D., Bailis, P., Gonzalez, J. E., Li, H., Zhang, Z., Franklin, M. J., Ghodsi, A., and Jordan, M. I. The missing piece in complex analytics: Low latency, scalable model management and serving with velox. In *CIDR*, 2015. URL http://cidrdb.org/cidr2015/Papers/CIDR15_Paper19u.pdf.
- Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., and Keogh, E. Querying and mining of time series data: Experimental comparison of representations and distance measures. *PVLDB*, 1(2):1542–1552, August 2008. ISSN 2150-8097. doi: 10.14778/1454159.1454226. URL <http://dx.doi.org/10.14778/1454159.1454226>.
- DiScala, M. and Abadi, D. J. Automatic generation of normalized relational schemas from nested key-value data. In *SIGMOD*, pp. 295–310, 2016. ISBN 978-1-4503-3531-7.
- Fernandez, R. C., Abedjan, Z., Madden, S., and Stonebraker, M. Towards large-scale data discovery: Position paper. In *ExploreDB*, pp. 3–5, 2016. ISBN 978-1-4503-4312-1.
- Fisher, R. A. On the probable error of a coefficient of correlation deduced from a small sample. *Metron*, 1: 3–32, 1921.
- Fisher, R. A. *Statistical Methods for Research Workers*, pp. 66–70. Springer-Verlag New York, 1992.
- Frazier, M., Goldman, S. A., Mishra, N., and Pitt, L. Learning from a consistently ignorant teacher. *J. Comput. Syst. Sci.*, 52(3):471–492, 1996. doi: 10.1006/jcss.1996.0035. URL <https://doi.org/10.1006/jcss.1996.0035>.
- Frigyik, B. A., Kapila, A., and Gupta, M. R. Introduction to the dirichlet distribution and related processes. Technical report, University of Washington Department of Electrical Engineering, 2010. UWEETR-2010-0006.
- Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572, 2014. URL <http://arxiv.org/abs/1412.6572>.
- Hynes, N., Scully, D., and Terry, M. The data linter: Lightweight, automated sanity checking for ml data sets. In *Workshop on ML Systems at NIPS 2017*, 2017.
- Khayyat, Z., Ilyas, I. F., Jindal, A., Madden, S., Ouzzani, M., Papotti, P., Quiané-Ruiz, J.-A., Tang, N., and Yin, S. Bigdansing: A system for big data cleansing. In *SIGMOD*, pp. 1215–1230, 2015.
- Krishnan, S., Franklin, M. J., Goldberg, K., and Wu, E. Boostclean: Automated error detection and repair for machine learning. *CoRR*, abs/1711.01299, 2017. URL <http://arxiv.org/abs/1711.01299>.
- Miller, B. P., Fredriksen, L., and So, B. An empirical study of the reliability of unix utilities. *Commun. ACM*, 33(12):32–44, December 1990. ISSN 0001-0782. doi: 10.1145/96267.96279. URL <http://doi.acm.org/10.1145/96267.96279>.
- Olston, C., Li, F., Harmsen, J., Soyke, J., Gorovoy, K., Lao, L., Fiedel, N., Ramesh, S., and Rajashekhar, V.

- Tensorflow-serving: Flexible, high-performance ml serving. In *Workshop on ML Systems at NIPS 2017*, 2017.
- Pearson, K. *On the Criterion that a Given System of Deviations from the Probable in the Case of a Correlated System of Variables is Such that it Can be Reasonably Supposed to have Arisen from Random Sampling*, pp. 11–28. Springer-Verlag New York, 1992.
- Pei, K., Cao, Y., Yang, J., and Jana, S. Deepxplore: Automated whitebox testing of deep learning systems. In *SOSP*, pp. 1–18, 2017. ISBN 978-1-4503-5085-3.
- Polyzotis, N., Roy, S., Whang, S. E., and Zinkevich, M. Data management challenges in production machine learning. In *SIGMOD*, pp. 1723–1726, 2017.
- Rekatsinas, T., Chu, X., Ilyas, I. F., and Ré, C. Holoclean: Holistic data repairs with probabilistic inference. *VLDB*, 10(11):1190–1201, August 2017. ISSN 2150-8097.
- Russell, S. and Norvig, P. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009. ISBN 0136042597, 9780136042594.
- Schelter, S., Boese, J.-H., Kirschnick, J., Klein, T., and Seufert, S. Automatically tracking metadata and provenance of machine learning experiments. In *Workshop on ML Systems at NIPS 2017*, 2017.
- Schelter, S., Lange, D., Schmidt, P., Celikel, M., Biessmann, F., and Grafberger, A. Automating large-scale data quality verification. *Proc. VLDB Endow.*, 11(12):1781–1794, August 2018. ISSN 2150-8097. doi: 10.14778/3229863.3229867. URL <https://doi.org/10.14778/3229863.3229867>.
- Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., and Dennison, D. Hidden technical debt in machine learning systems. In *NIPS*, pp. 2503–2511, 2015. URL <http://dl.acm.org/citation.cfm?id=2969442.2969519>.
- Sparks, E. R., Venkataraman, S., Kaftan, T., Franklin, M. J., and Recht, B. Keystoneml: Optimizing pipelines for large-scale advanced analytics. In *ICDE*, pp. 535–546, 2017.
- Stonebraker, M., Bruckner, D., Ilyas, I. F., Beskales, G., Cherniack, M., Zdonik, S. B., Pagan, A., and Xu, S. Data curation at scale: The data tamer system. In *CIDR*, 2013.
- Vartak, M. MODELDB: A system for machine learning model management. In *CIDR*, 2017.
- Volkovs, M., Chiang, F., Szlichta, J., and Miller, R. J. Continuous data cleaning. In *ICDE*, pp. 244–255, 2014. doi: 10.1109/ICDE.2014.6816655.
- Wang, X., Dong, X. L., and Meliou, A. Data x-ray: A diagnostic tool for data errors. In *SIGMOD*, pp. 1231–1245, 2015. ISBN 978-1-4503-2758-9. doi: 10.1145/2723372.2750549. URL <http://doi.acm.org/10.1145/2723372.2750549>.
- Witten, I. H., Frank, E., and Hall, M. A. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011. ISBN 0123748569, 9780123748560.

A CONSTRAINTS IN THE DATA SCHEMA

```
message Feature {
  ...
  // Limits the distribution drift between training
  // and serving data.
  FeatureComparator skew_comparator;

  // Limits the distribution drift between two
  // consecutive batches of data.
  FeatureComparator drift_comparator;
}
```

Figure 7: Extensions to the Feature message of Schema to check for distribution drifts.

We explain some of these feature level characteristics below using an instance of the schema shown in Figure 3:

Feature type: One of the key invariants of a feature is its data type. For example, a change in the data type from integer to string can easily cause the trainer to fail and is therefore considered a serious *anomaly*. Our schema allows specification of feature types as INT, FLOAT, and BYTES, which are the allowed types in the `tf.train.Example` (`tfe`) format. In Figure 3, the feature “event” is marked as of type BYTES. Note that features may have richer semantic types, which we capture in a different part of the schema (explained later).

Feature presence: While some features are expected to be present in all examples, others may only be expected in a fraction of the examples. The FeaturePresence field can be used to specify this expectation of presence. It allows specification of a lower limit on the fraction of examples that the feature must be present. For instance, the property `presence: {min_fraction: 1}` for the “event” feature in Figure 3 indicates that this feature is expected to be present in all examples.

Feature value count: Features can be single valued or lists. Furthermore, for features that are lists, they may or may not all be of the same length. These value counts are important to determine how the values can be encoded into the low-level tensor representation. The ValueCount field in the schema can be used to express such properties. In the example in Figure 3, the feature “event” is indicated to be a scalar as expressed using the min and max values set to 1.

Feature domains: While some features may not have a restricted domain (for example, a feature for “user queries”), many features assume values only from a limited domain. Furthermore, there may be related features that assume values from the same domain. For instance, it makes sense for two features like “apps_installed” and “apps_used” to be drawn from the same set of values. Our schema allows specification of domains both at the level of individual features as well as at the level of the schema. The named

domains at the level of schema can be shared by all relevant features. Currently, our schema only supports shared domains for features with string domains.

A domain can also encode the semantic type of the data, which can be different than the raw type captured by the TYPE field. For instance, a bytes features may use the values “TRUE” or “FALSE” to essentially encode a boolean feature. Another example is an integer feature encoding categorical ids (e.g., enum values). Yet another example is a bytes feature that encodes numbers (e.g., values of the sort “123”). These patterns are fairly common in production and reflect common practices in translating structured data into the flat `tf.train.Example` format. These semantic properties are important for both data validation and understanding, and so we allow them to be marked explicitly in the domain construct of each feature.

Feature life cycle: The feature set used in a machine learning pipeline keeps evolving. For instance, initially a feature may be introduced only for experimentation. After sufficient trials, it may be promoted to a *beta* stage before finally getting upgraded to be a *production* feature. The gravity of anomalies in the features at different stages is different. Our schema allows tagging of features with the stage of life cycle that they are currently in. The current set of stages supported are UNKNOWN_STAGE, PLANNED, ALPHA, BETA, PRODUCTION, DEPRECATED, and DEBUG_ONLY

Figure 2 only shows only a fragment of the constraints that can be expressed by our schema. For instance, our schema can encode how groups of features can encode logical sequences (e.g., the sequence of queries issued by a user where each query can be described with a set of features), or can express constraints on the distribution of values over the feature’s domain. We will cover some of these extensions in Section 4, but we omit a full presentation of the schema in the interest of space.

B STATISTICAL SIGNIFICANCE OF MEASUREMENTS OF DRIFT

Suppose that we have two days of data, and we have some measure of their distance? As we discussed in Section 4, this measure will never be zero, as some real drift is always expected. However, how do we know if we can trust such a measurement?

Specifically, suppose that there is a set S where $|S| = n$ of distinct observations we can make, and $\Delta(S)$ is the set of all distributions over S . Without loss of generality, we assume $S = \{1 \dots n\}$. In one dataset, there are m_p observations $P_1 \dots P_{m_p} \in S$ with an empirical distribution $\hat{p} = \hat{p}_1 \dots \hat{p}_n$. In a second dataset, there are

m_q observations $Q_1 \dots Q_{m_q}$ with an empirical distribution $\hat{q} = \hat{q}_1 \dots \hat{q}_n$. We can assume that the elements in $P_1 \dots P_{m_p}$ were drawn independently from some distribution $p = p_1 \dots p_n$, that was in turn drawn from a fixed Dirichlet prior (Frigyik et al., 2010) $Dir(\alpha)$ where $\alpha = (1 \dots 1)$ (a uniform density over the simplex), and similarly, $Q_1 \dots Q_{m_q}$ were drawn independently from some distribution $q = q_1 \dots q_n$, that was in turn drawn from the same fixed Dirichlet prior.

If we have a particular measure $d : \Delta(S) \times \Delta(S) \rightarrow \mathbf{R}$, we have two options. First, we could measure the *empirical drift* $d(\hat{p}, \hat{q})$. Or we could attempt to estimate the *theoretical drift* $d(p, q)$. Although p and q are not directly observed, the latter can be achieved more easily than one might expect. First of all, the posterior distribution over p given the observations $P_1 \dots P_{m_p}$ is simply $Dir(\alpha + m_p \hat{p})$, and the posterior distribution of q is $Dir(\alpha + m_q \hat{q})$ (see Section 1.2 in (Frigyik et al., 2010)). Thus, one can get a sample from the posterior distribution of $d(p, q)$ by sampling p' from $Dir(\alpha + m_p \hat{p})$ (see Section 2.2 in (Frigyik et al., 2010)) and q' from $Dir(\alpha + m_q \hat{q})$ and calculating $d(p', q')$.

One metric we use is $d_1(p, q) = \sum_{i=1}^n |p_i - q_i|$. The advantage of d_1 is that it corresponds to how visually distinguishable two distributions are (see Figure 8). Consider a field that has 100 values that are all equally likely. If all values are uppercase instead of lowercase, then the $d_\infty(p, q) = 0.01$, but $d_1(p, q) = 1$ (the highest possible value).

What we find, if estimate the theoretical drift as described above, is that when m_p and m_q are at least a billion and $n < 100$, then $d(p, q)$ is very close to $d(\hat{p}, \hat{q})$. In other words, for the datasets that we care about, the empirical drift $d_\infty(\hat{p}, \hat{q})$ is a sound approximation of the theoretical drift.

Moreover, the empirical drift of d_1 can be visualized:

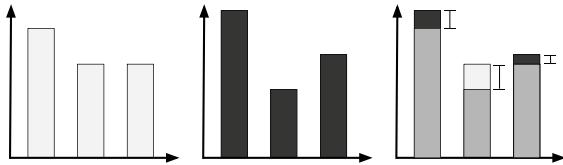


Figure 8: Two distributions, white and black are compared. When overlaying them, the difference can be “seen”. The sum of the magnitude of these visible differences is the d_1 distance.

The connection to visualization is important: for instance, if we observe a high KL divergence, Kolmogorov-Smirnov statistic, or a low cosine similarity, it is likely that the first thing a human will do is look at the two distributions and visually inspect them. The scenarios where the L1 distance is low but the KL divergence is high correspond to when

very small probabilities (10^{-6}) become less small (10^{-3}). It is unclear whether such small fluctuations on the probability of rare features are cause for concern in general, whereas the magnitude of d_1 directly corresponds to the number of examples impacted, which in turn can affect performance.



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Analytics

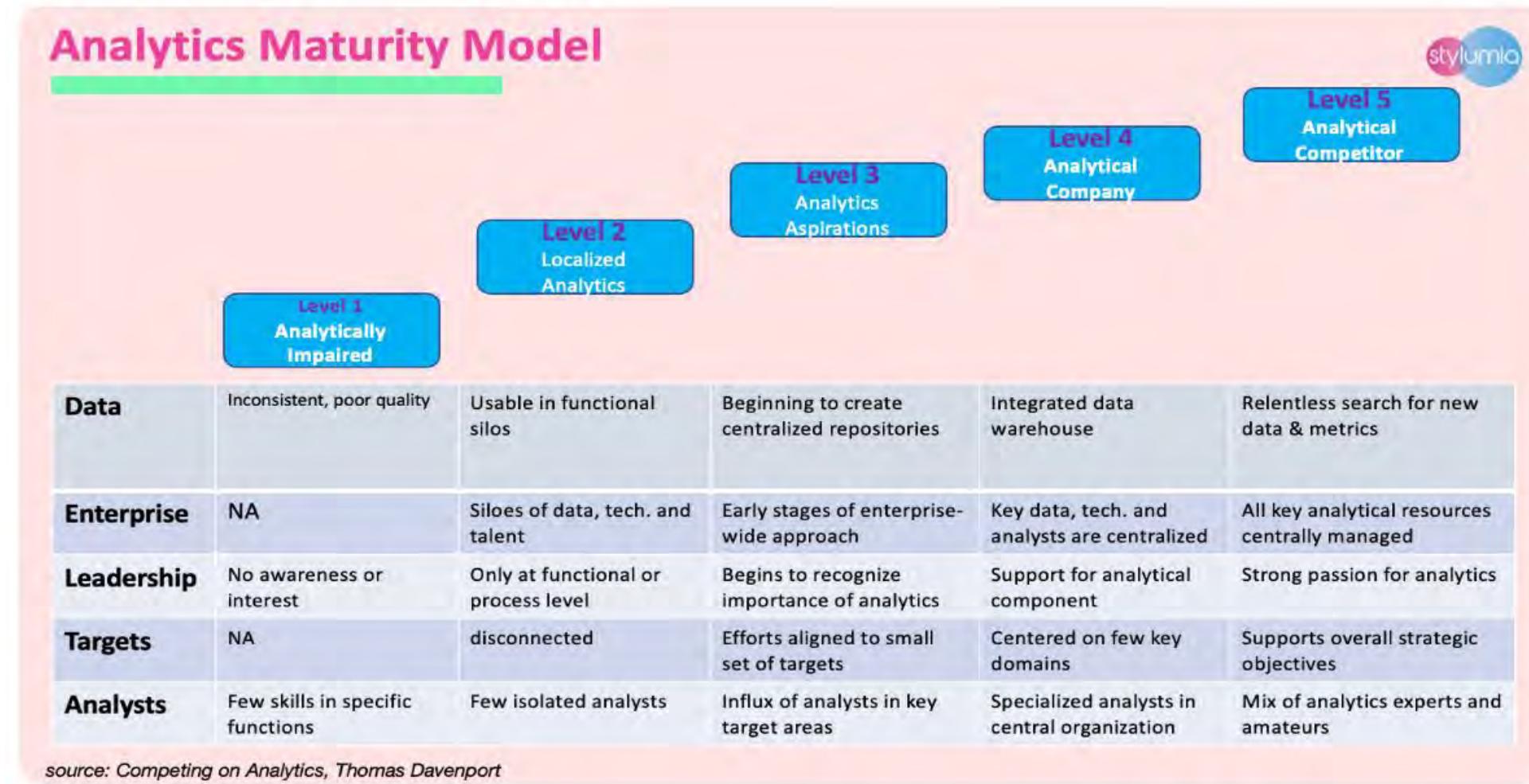
Pravin Y Pawar

Analytics

Defined

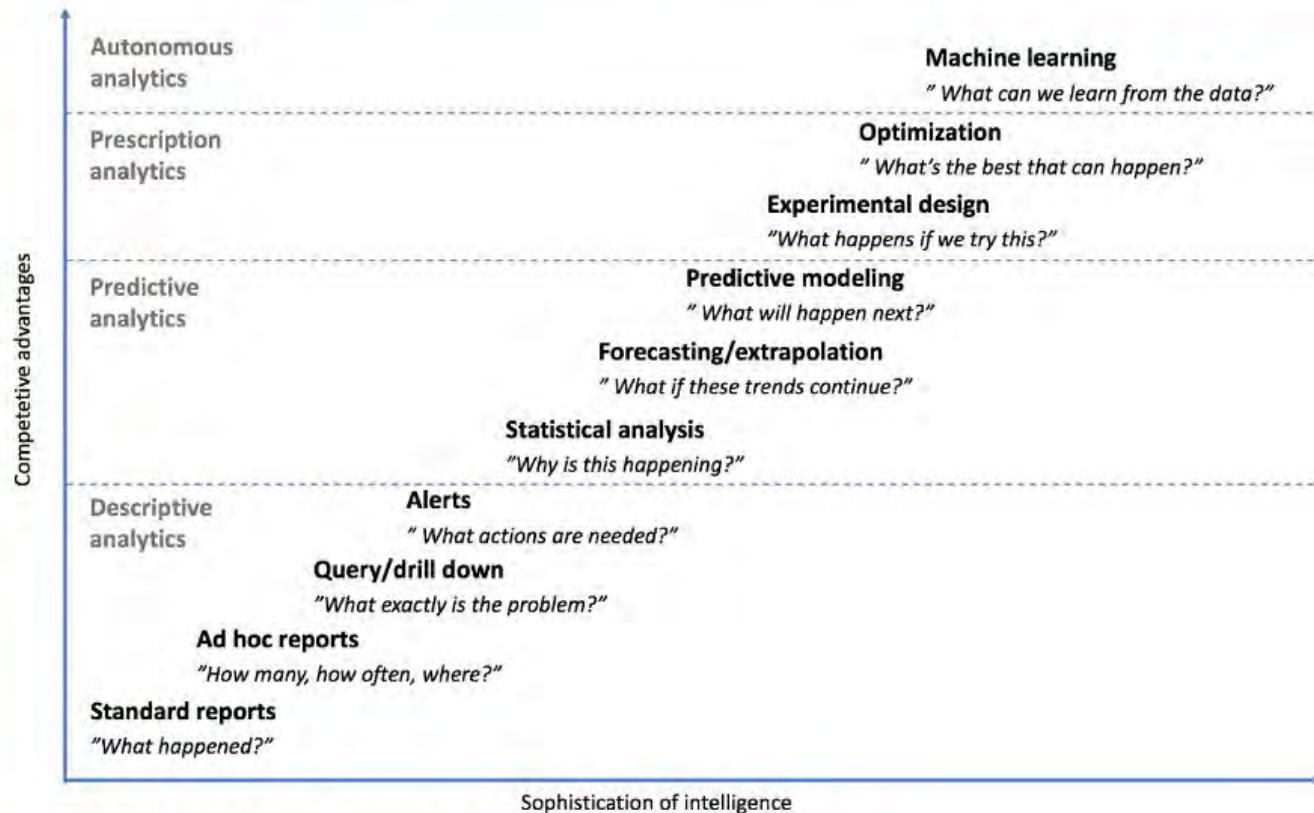
- Extensive use of data, statistical and quantitative analysis, explanatory and predictive models, and fact based management to drive decisions and actions
- Purpose
 - ✓ To unearth hidden patterns
 - ✓ To decipher unknown correlations
 - ✓ Understand the rationale behind trends
 - ✓ Mine useful business information
- Helps in
 - ✓ Effective marketing
 - ✓ Better customer service and satisfaction
 - ✓ Improved operational efficiency
 - ✓ Competitive advantage over rivals

Analytics Maturity Model



Analytics Maturity and Competitive Advantage

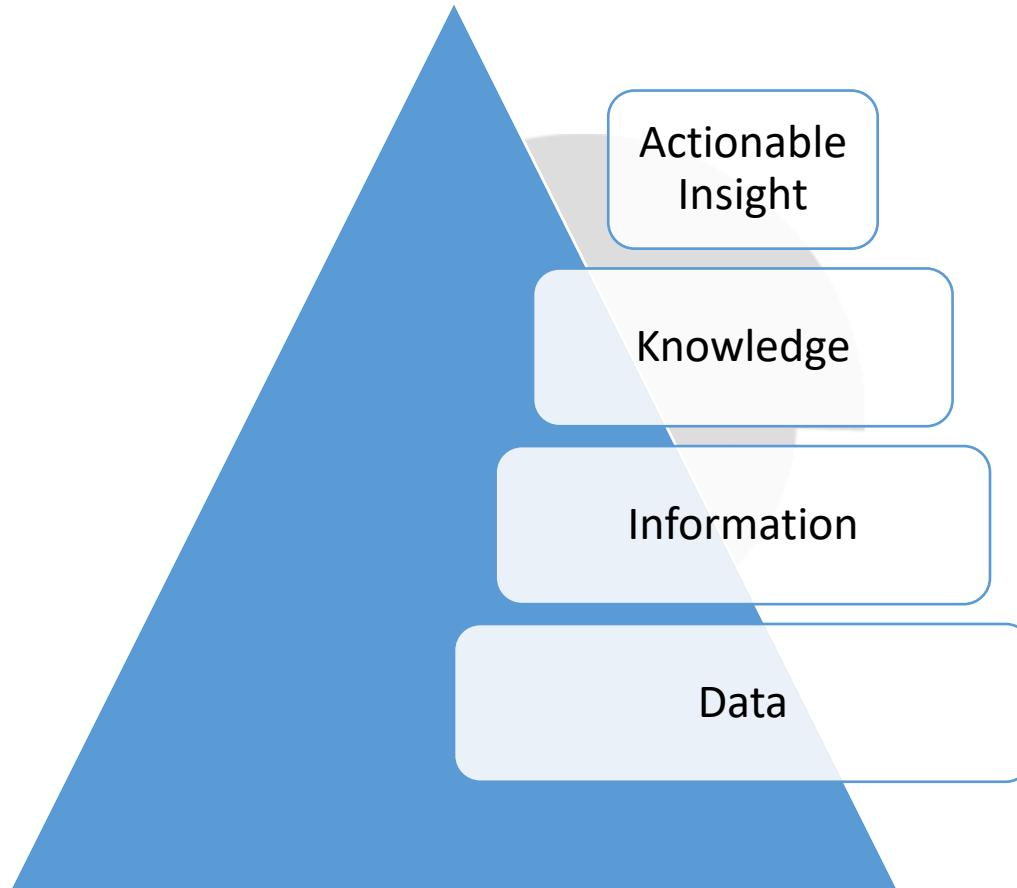
Analytics Maturity & Competitive Advantage



source: Competing on Analytics, Thomas Davenport

Process of Analysis

Transformation of Data



Types of Analytics (1)

Descriptive Analytics

- aka Business Intelligence (BI) or Performance reporting
- Provides access to historical and current data
- Provides ability to alert, explore and report using internal and external data
- Reports on events , occurrences of the past
- Usually data from legacy systems used for analysis
- Based on relational databases
- Sometimes also referred as Analytics 1.0
- Era : mid 1950s to 2009
- Questions asked
 - ✓ What happened?
 - ✓ Why did it happen?

Types of Analytics (2)

Predictive Analytics

- Uses quantitative techniques like segmentation, forecasting etc. but also makes use of descriptive analytics for data exploration
- Era : from 2005 to 2012
- Uses technologies like models and rule based systems
- Uses past data to predict the future
- Based on huge data gathered over period of time
- Externally sourced data also used
- aka Analytics 2.0
- Key questions
 - ✓ What will happen?
 - ✓ Why it will happen?

Types of Analytics (3)

- Prescriptive
 - ✓ Uses a variety of quantitative techniques like optimization and technologies like models, machine learning and recommendations engines
 - ✓ Suggests optimal behaviors and actions
 - ✓ Aka Analytics 3.0 = Descriptive + Predictive + Prescriptive
 - ✓ Uses data from past to make prophecies of future and at the same time make recommendations to leverage the situation to one's advantage
 - ✓ Data is blend from Big data and legacy systems , ERP, CRM etc.
- Autonomous
 - ✓ Uses AI or cognitive technologies to create and improve models and learn from data
 - ✓ All without human hypothesis and with less involvement by human analysis

Another thought

- Basic Analytics
 - ✓ Slicing and dicing of data to help with basic insights
 - ✓ Reporting on historical data, basic visualizations etc.
- Operationalized Analytics
 - ✓ Analytics integrated in business processes
- Advanced Analytics
 - ✓ Forecasting the future by predictive modelling



Thank You!

In our next session: Big Data Analytics



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Data Integration

Pravin Y Pawar

Adapted from [Streamset's Data Integration guide](#)

Data integration

Meaning

- Data integration **combines various types and formats of data from any source across an organization into a data lake or data warehouse**
 - to provide a unified fact base for analytics
- Unified datasets allows businesses to
 - make better decisions,
 - aligns departments to work better together,
 - and drives better customer experience
- Means **consolidating data from multiple sources into a single dataset**
 - to be used for consistent business intelligence or analytics
- Very **simple explanation** for a complex topic that has evolved over its **30 year history**
 - data integration has **transitioned from a backend, retrospective process into core real-time infrastructure**

Data integration(2)

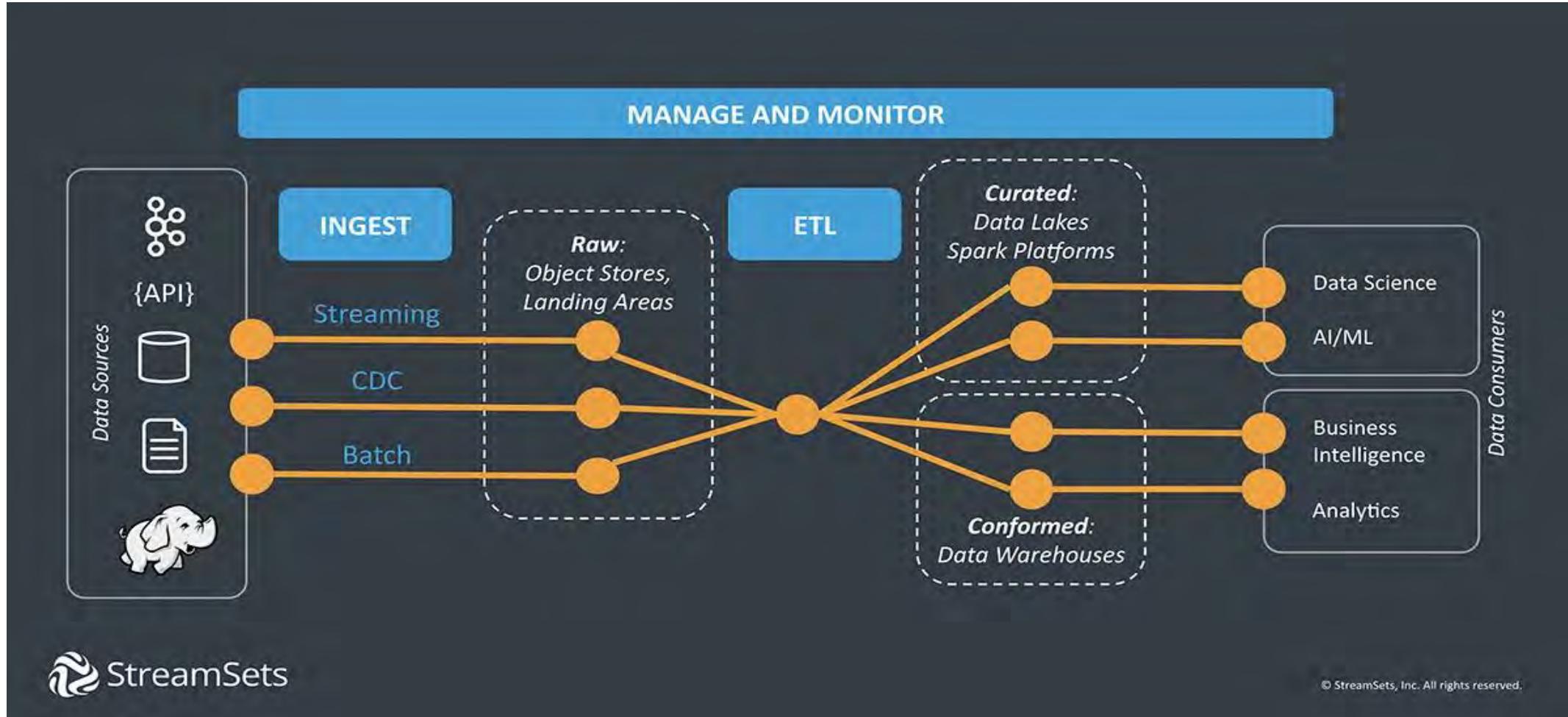
Working

- To move data from one system to another requires a data pipeline that
 - understands the structure and meaning of the data
 - as well as defines the path it will take through the technical systems
- A relatively simple and common type of data integration is data ingestion,
 - where data from one system is integrated on a regular basis into another system
- More complex way
 - may also include cleansing, sorting, enrichment and other processes to make the data ready for use at its final destination
 - Sometimes this happens before the data is stored and the process is called ETL (extract, transform, load)
 - Other times it makes more sense to store the data first, then prepare it for use known as ELT (extract, load, transform)

Data Integration Evolution

- In the early 1990s, when companies began adopting data warehouses to collect data from multiple systems to fuel analysis,
 - there were no smartphones or ecommerce
 - Salesforce and Software as a Service as a category did not yet exist
 - Amazon had not sold a single book, much less on-demand computing
- Back then:
 - Data came **from business applications and operational databases in a structured format** that could be mapped to the structure required for analysis
 - Data arrived and was **processed in batches**, creating snapshots of the business in time and **stored in data warehouses or data marts**
 - Data was used for **financial reporting, sales dashboards, supply chain analytics**, and other essential functions of the enterprise
- Data integration was **primarily the responsibility of ETL developers**,
 - who used **hand coding or specialized software** to create ETL mappings and jobs
 - developed **specialized skills** related to the **source and target systems** they integrated
- Data integration was **owned and governed by enterprise IT**
 - with control of the **hardware and software used to collect data, store it, and analyze it**
 - focused on **performance, security, and cost** of the monolithic data management systems

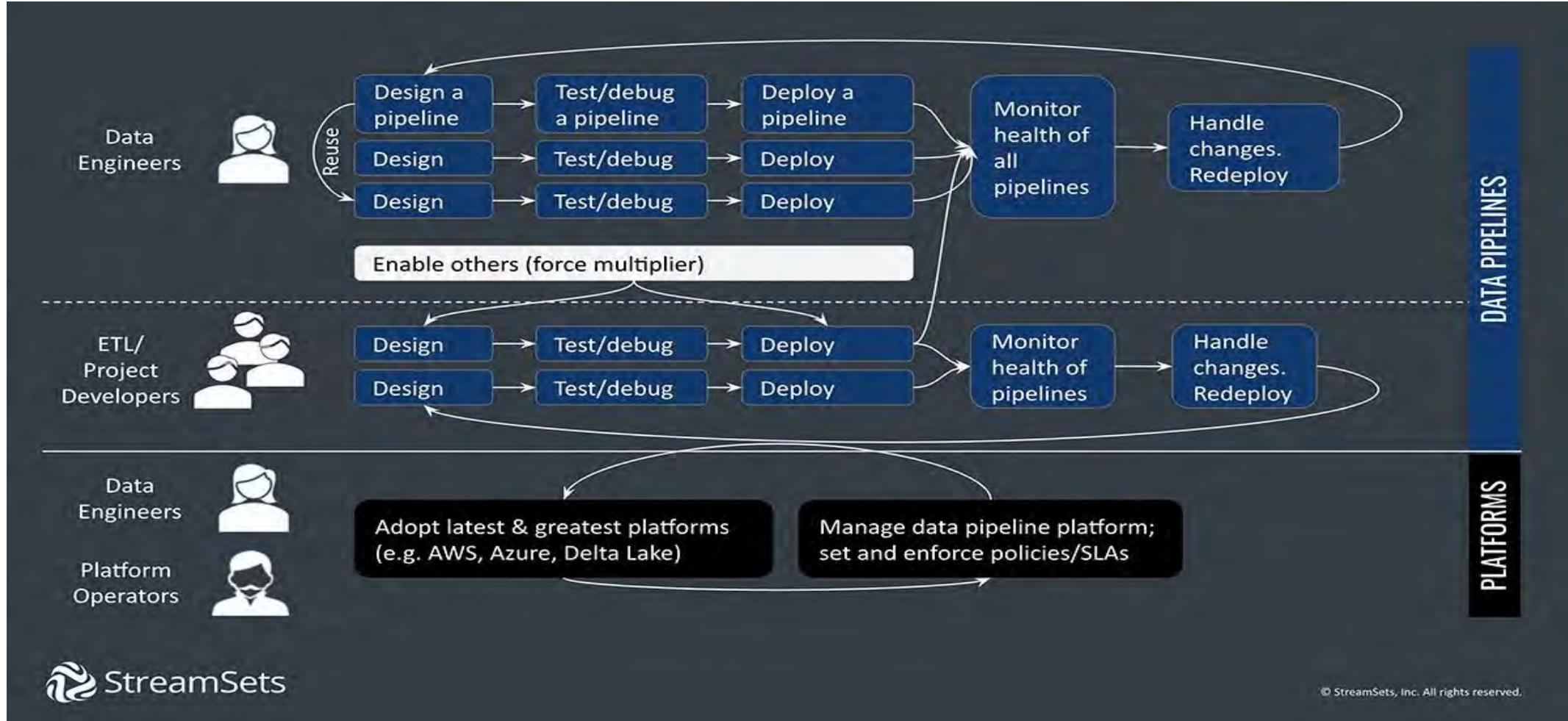
Modern Data Integration Space



Modern Data Integration Challenges

- The world according to data looks very different today!
- Enterprise data integration have been transformed by
 - explosion of data, data sources (IoT, APIs, cloud applications, on premise data, various databases, and more)
 - data structures combined with radical innovation in infrastructure services, compute power, analytic tools and machine learning
- Demand for (continuous) data
 - Real-time decision making and real-time services require continuous data that is transformed in flight
 - DevOps and agile software development practices have spread throughout the organization
 - increasing the demand for always on, self-service data
 - The move from on-premises to the cloud for applications as well as computing services requires cloud data integration,
 - data integration beyond the walled garden of the corporate data center
- Suddenly, the full lifecycle of data integration matters as much as the initial implementation
 - has to support continuous integration of data from different sources, and continuous data delivery as well as continuous innovation,
 - and that takes automation
 - focus is not just on the “how” of implementation, but on “what” is needed by the business

New Roles and New Responsibilities



Data Integration Tools

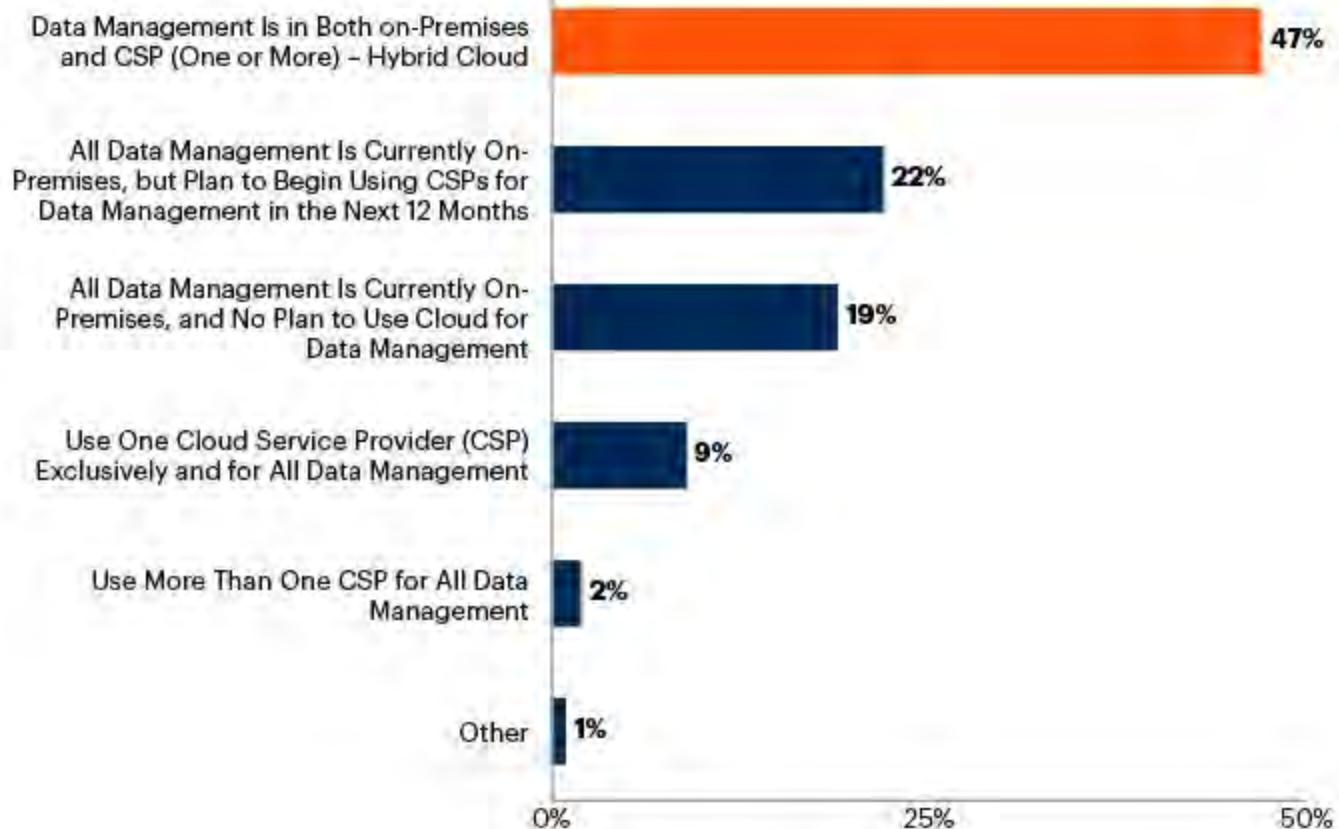
- Data Integration tools are software-based tools that ingest, consolidate, transform, and move data from source(s) to destination,
 - performing mappings, transformations, and data cleansing along the way
- Ultimately, they integrate the data into a ‘single source of truth’ destination,
 - such as a data lake or data warehouse
 - allows consistent, reliable data for use in analytics and business intelligence

Data Integration Tool Considerations for the Age of Data Engineering

- When choosing a tool for data integration, there are a few important considerations to take into account:
- What **type of data** will be in your data pipeline?
 - Structured,
 - Unstructured
 - and Semi-structured Data
- How will that data be **processed**?
 - Batch,
 - Micro-batch,
 - and Stream Processing
- Where will the **data come from and go to**?
 - On-premises,
 - Cloud,
 - Multi-cloud,
 - and Hybrid Architectures

Moving to Hybrid Cloud

Use of Hybrid Cloud
Percentage of Respondents



n = 129

Base: Gartner Research Circle Members. Excludes "Not sure."

Q: Which best describes how your organization is using the cloud for Data Management?

Source: Gartner



Thank You!

In our next session:



Magic Quadrant for Data Integration Tools

Published 17 August 2022 - ID G00758102 - 118 min read

By Ehtisham Zaidi, Sharat Menon, [and 2 more](#)

The data integration tools market is seeing renewed momentum driven by requirements for multicloud and hybrid data integration and data fabric design patterns. Data and analytics leaders should use this research to evaluate suitable vendors for their existing and upcoming data integration use cases.

Strategic Planning Assumptions

- Through 2024, manual data integration tasks will be reduced by up to 50% through the adoption of data fabric design patterns that support augmented data integration.
- By 2024, AI-enabled augmented data management and integration will reduce the need for IT specialists by up to 30%.
- By 2025, data integration tools that do not provide capabilities for multicloud hybrid data integration through a PaaS model will lose 50% of their market share to those vendors that do.

Market Definition/Description

Gartner defines data integration as the discipline comprising the architectural patterns, methodologies and tools that allow organizations to achieve consistent access and delivery of data across a wide spectrum of data sources and data types to meet the data consumption requirements of business applications and end users. Data integration tools enable organizations to access, integrate, transform, process and move data spanning various endpoints and across any infrastructure to support their data integration use cases.

The market for data integration tools includes vendors that offer a stand-alone software product (or products) to enable the construction and implementation of data access and data delivery infrastructure for a variety of data integration use cases.

These include (but are not limited to):

- **Data engineering:** The usage of data integration tool capabilities to engineer data pipelines in support of various analytical use cases such as data warehouse, data lakes, data science and machine learning. We are not referring to data engineering as a practice here.
- **Cloud data integration:** Migrating and modernizing data workloads in the public cloud with an architecture that spans on-premises and one or more cloud ecosystems (hybrid/multicloud) to enable an optimal use of cloud resources.
- **Operational data integration:** Supporting operational/transactional data integration use cases such as master data management (MDM), interenterprise data acquisition and sharing, B2B data sharing, synchronizing data related to critical business processes, and supporting data governance initiatives.
- **Data fabric:** Data integration capabilities delivered in support of use cases related to the emerging data fabric design. This includes the ability to enable faster access to trusted data across distributed landscapes by utilizing active metadata, semantics and ML capabilities.

The core capabilities are functional requirements that each evaluated data integration tool vendor must support to be included in this research.

These include:

- **Data movement topology:** Uni-/bi-/multi-directional movement of data across endpoints (e.g., synchronize, compare, broadcast, consolidate) via physical and virtual modes, meeting batch/microbatch/real-time latency requirements for data integration.
- **Data virtualization:** Executing distributed queries against disparate data sources that are virtually integrated. This requires adapters to data sources, a metadata repository and a distributed query engine that can provide results in various ways (e.g., API, JDBC) for downstream consumption.

- **Stream data integration:** Processing data in motion (e.g., streams, events) and provisioning the in-stream data for downstream consumption, analysis or storage.
- **API services:** Data-as-a-service enabled through API design capabilities to create and manage outbound API endpoints over existing data assets and handle inbound API consumption to ingest internal and external data.
- **Complex data transformation:** Capabilities that ease complex data processing operations such as fixing outliers, sophisticated parsing (e.g., free-form text mining, telemetry logs, media mining), complex data modeling (e.g., automated data pipeline creation and data warehouse automation support) and creating reusable transformations.
- **Augmented data integration:** Capabilities that improve and optimize data integration operations (e.g., self-healing schema drifts, autorecovery) using extensive use of metadata (e.g., usage data, transaction logs, system workloads) and prepackaged ML algorithms that can inform or automate the tasks to ingest, transform, combine and provision data.
- **Data preparation:** The suitability of data integration tools to support a range of business roles (e.g., citizen integrators, business analysts) for self-service data integration. The emphasis is on empowering nontechnical staff using various techniques such as low-/no-code data blending, visual exploration and probabilistic matching.
- **Integration portability:** Data flow design portability across infrastructure (on-premises, SaaS, cloud service provider, VPC, etc.), providing workload management capabilities in a clean, safe and portable runtime environment (e.g., through containerization).

The support capabilities consist of those features that support forward-looking use cases through differentiated functionality.

These include:

- **Metadata support:** Capabilities that support the extensive use of metadata (e.g., usage metadata, transaction logs, system workloads) to automate/improve data integration tasks.
- **Data governance support:** Capabilities that assist data governance mandates (e.g., data quality, data lineage) while handling data for meeting specific data integration use cases (e.g., MDM, data sharing)
- **DataOps support:** Change management capabilities to support data and related artifacts (e.g., Git integration of data pipelines, data model management), automation (e.g., automated testing), orchestration of data delivery (e.g., CI/CD pipelines) with appropriate

levels of security to improve the use and value of data.

- **FinOps support:** Capabilities that enable data and analytics leaders to iteratively control spending, understand product performance and make choices regarding price-to-performance trade-offs, resulting in optimal allocation of resources in the cloud.

For a vendor to be included in this market, its data integration tools must be able to support these use cases and capabilities **independent** of other vendor's product offerings. Vendors that sell data integration technology as part of other solutions (such as analytics platforms, DBMSs, and packaged or SaaS applications) are not considered data integration tool vendors by Gartner.

Our evaluation of data integration tools does not include open-source frameworks, general-purpose development platforms or programming interfaces. Such data integration frameworks or platforms that require heavy customization by developers to engineer them for specific data integration scenarios are excluded from this Magic Quadrant.

Vendors evaluated in this Magic Quadrant offer at least one commercial off-the-shelf tool that is purpose-built for supporting all the listed data integration use cases in this report.

Magic Quadrant

Figure 1: Magic Quadrant for Data Integration Tools





Source: Gartner (August 2022)

Vendor Strengths and Cautions

Amazon Web Services

Amazon Web Services (AWS) is a Niche Player in this Magic Quadrant; it is a new entrant this year. It is headquartered in Seattle, Washington. It offers AWS Glue as its data integration tool. Its customers use AWS Glue when their primary target data stores reside on the AWS cloud. Gartner estimates thousands of customers using AWS Glue data integration jobs and catalog service. Its operations are global and across various sectors.

Strengths

- **Native integration reduces complexity and improves interoperability within the AWS data ecosystem:** AWS Glue's native integration with other AWS cloud services (like S3, Redshift, Athena and Lake Formation) makes it the most suitable data integration tool choice when selecting an AWS-native data ecosystem.
- **Innovation to support data delivery demands:** AWS Glue scores well on innovation for its serverless data integration service that goes beyond scaling infrastructure for data processing. It supports ML-based data cleansing and PII detection at scale. It even extends data processing for other applications (such as Notebooks and IDEs) through its interactive sessions API.
- **Centralizes metadata in its data catalog:** AWS Glue offers a rich set of metadata capabilities such as cataloging, search, lineage, data quality checks, PII detection and data access logging for auditability. For example, AWS Glue metadata can be accessed within Athena for exploring the underlying data in S3.

Cautions

- **Limited connectors beyond AWS-native services:** AWS Glue has a wide range of native connectors from the AWS marketplace for on-premises and other clouds. However, for certain non-AWS sources (like Mainframes, SAP, Azure or GCP) customers will need third-party options, some of which are available from AWS Marketplace (like CData, tCVision and Qlik) to compensate for this deficit.
- **Steep learning curve:** AWS Glue is not friendly to beginners — according to Gartner Peer Insights reviews, its product documentation and sample code libraries are not enough for complex data engineering tasks. There is a high reliance on technical expertise for coding (Python or Scala) and debugging, and implementing an enterprise integration framework requires expertise in serverless architectures. In addition to recent product improvements, AWS Glue has added no-code tools like Glue Studio and Glue Databrew to mitigate these challenges.

- **Practitioners report several operational issues:** Gartner Peer Insights reviews highlight several operational issues such as high startup times for cluster spin and new jobs, lack of idle timeout settings leading to high costs, and occasional performance bottlenecks during peak hours of data processing. However, the latest AWS Glue 3.0 release should alleviate most of these issues.

CloverDX

CloverDX is a Niche Player in this Magic Quadrant, the same as last year. It is headquartered in Prague, Czech Republic, and offers the CloverDX Data Management Platform, composed of CloverDX Designer and CloverDX Server. The customer base includes more than 450 organizations, mainly located in North America and Europe. Clients are commonly in the following sectors: finance and banking, software and SaaS providers, advisory and consulting, and the public sector.

Strengths

- **Broad support for complex data types and data integration scenarios:** CloverDX supports a wide variety of data types and also has robust data transformation capabilities. It supports CSV, XML, JSON and Avro data types and beyond, offering connectivity to relational datastores. It also integrates with graph and multimodel databases such as Neo4j and CosmosDB (among several others). CloverDX has also incorporated built-in support for Kafka.
- **Simple pricing model:** Although CloverDX does not have particularly strong support for financial governance and FinOps (it does not offer ways to automatically scale deployments), the platform has a pricing model that is easy to understand. Pricing is based on user seats and server cores, so costs are transparent and predictable. This is in contrast to the consumption-based pricing models used by other vendors, which can result in unplanned or unpredictable costs.
- **Customer experience:** Clients praise CloverDX's high-touch customer support as well as its remote and self-paced training options and personalized professional services. They report that they are able to quickly find value from the product. The company has a customer advisory board that helps drive the product development roadmap, demonstrating that it is attuned to user needs.

Cautions

- **Metadata, data fabric and automation capabilities are still developing:** As a company, CloverDX understands the common complexities of the data landscape and recognizes the need to offer automation for data integration. Although the CloverDX Data Management Platform does provide automated capabilities, fully executing on this vision is a work in progress. As an example,

CloverDX does not have strong support for the continuous analysis of metadata (also referred to as “active metadata”), which is important for setting the foundation for a data fabric and for additional automation.

- **Limited support for diverse data integration styles:** CloverDX does not directly support stream data integration, such as modifying data in-stream for the next consumer, although this work can usually be achieved by microbatching. CloverDX also does not support data virtualization or change data capture.
- **Limited third-party partnerships:** CloverDX is working to expand its network of integration partners (with system integrators and OEMs, for example), but currently lags behind its larger competitors in terms of number and breadth of partnerships. CloverDX needs to improve integration with adjacent data management infrastructure providers such as cloud DBMSs, external and third-party metadata management tools, data governance technology providers, and analytics and data science providers.

Denodo

Denodo is a Leader in this Magic Quadrant. In the previous iteration of this research, it was also a Leader. It is based in Palo Alto, California. It offers Denodo Platform on-premises and on public cloud (AWS, Azure and GCP). Its operations are geographically diverse, with more than 1,000 customers primarily in the financial services, manufacturing and technology sectors.

Strengths

- **Focus on distributed data architectures:** Denodo begins with virtualization across disparate sources and extends its capabilities through data science notebooks. It can write out its caching tier to persistent stores. It is suitable for logical data warehouse and data fabric use cases through building business-friendly semantic models.
- **Sophisticated optimizer:** Denodo uses statistics to evaluate current operations of query patterns and then uses ML-enabled DataOps to enhance performance in terms of faster response and smaller footprint in terms of resource allocation.
- **“Try before you buy” engagement model:** Denodo’s customers appreciate its presales support and proof-of-concept activities. About 80% of its paying customers have tried Denodo Express (its free product, which is capped at a certain capacity for a single user with standard features) at some stage in their sales cycle. Its pricing model is based on the number of cores driven by customer demand, varying from small departmental to enterprise deployments.

Cautions

- **Limited support for traditional bulk/batch workloads:** Denodo does not support change data capture within its current data integration offering. It is therefore not suited for traditional batch-related extract, transform and load (ETL) operations, given its focus on federation and distributed query processing.
- **Multilocation deployments require manual configuration:** Denodo supports hybrid and intercloud data integration use cases. However, connectivity across Denodo instances running in different geographies involves a lot of manual configurations (particularly in Denodo version 7.x) and ongoing operations support to ensure effectiveness of the multilocation deployment. Denodo claims that it has addressed this gap in its latest version (8.x), which requires customers to upgrade.
- **Data security configuration can be challenging:** Denodo practitioners report challenges around the process of secure authentication configuration, SSL connections setup in cloud environments and frequent timeouts. A few customers have complained of delays in the platform readiness because some configuration settings were difficult to manage when done through scripting instead of using the Denodo UI.

Fivetran

Fivetran is a Niche Player in this Magic Quadrant. In the previous iteration of this research, it was also a Niche Player. It is headquartered in Oakland, California. It acquired HVR in October 2021. It offers the following data integration tools: Fivetran and HVR. It has more than 4,000 customers for these products. Its operations are primarily focused in North America and EMEA, with a growing presence in APAC. The top three industries it supports are in the software services, information technology and media sectors.

Strengths

- **Market momentum and partnerships:** Fivetran grew its customer base significantly last year, primarily targeting departmental leaders with its appeal of easy, automated extract and load data integration capabilities. It has established partnerships with major cloud service providers (CSPs), independent software vendors (ISVs) and system integrators (SIs) to further strengthen its market momentum.
- **Product strategy focused on data movement and cloud:** Fivetran focuses on physical data movement into cloud targets, making it suitable for use cases like centralizing data in cloud data stores, database replication and cloud migration. It has made several improvements to its integration infrastructure compliance and secure data movement such as secure data encryption, expanded RBAC and data residency controls, and private links (with AWS and Azure).

- **Low total cost of ownership (TCO) with quick data delivery:** Fivetran applies a consumption-based pricing model to the amount of monthly active rows processed, allowing customers to get started with their projects without the need to secure significant upfront capital expenditure. It also reduces data pipeline development times and minimizes operations. It can be set up to handle source schema changes automatically in the cloud target data store by managing schema drifts and guaranteeing reliable data replication. As a result, its customers spend less time on data operations tasks.

Cautions

- **Limited product coverage:** Fivetran offers low-latency, managed data ingestion service to the cloud. However, traditional batch-related ETL operations, data virtualization and message-oriented middleware are not supported. As a result, enterprise customers need to supplement Fivetran's data ingestion capabilities with a third-party technology if they require complex data transformation and orchestration support. Fivetran integrates with dbt Core to enable data transformation for its customers.
- **Selective metadata and governance support:** Fivetran lacks the ability for metadata analysis like discerning relationships between data assets, profiling and dynamically alerting discrepancies. It is beginning to open up its metadata via API to third-party catalogs. It also lacks essential data governance capabilities in support of data quality and enforcement of compliance rules like data masking on integrated data. Its roadmap item "governance for inflight data" is likely to introduce automatic PII detection and contextual tagging.
- **Customer support and operational issues:** The poor user interface inherited from the HVR acquisition, combined with challenges around locating suitable knowledge base documentation often forces Fivetran's HVR customers to seek product support. This challenge has created longer turnaround times for some customers. The latest release of HVR 6 with updated documentation and recent advances with Fivetran's support capabilities are expected to alleviate these challenges.

Hitachi Vantara

Hitachi Vantara has reentered the Magic Quadrant this year after its absence last year and is positioned as a Niche Player. It is based in Santa Clara, California, and offers Lumada DataOps platform, which includes Data Integration & Analytics powered by Pentaho and Data Catalog. It also offers data integration capabilities within Lumada Industrial DataOps, its industrial IoT (IIoT) platform. The vendor's operations are geographically diverse with a customer base of around 800 organizations. The top three industries it supports are financial services, software and technology, and retail and consumer products.

Strengths

- **Improved data management vision:** Lumada DataOps provides data integration capabilities with support for adjacent data management disciplines, such as data governance, metadata management and data quality, through the acquisitions of Waterline Data and Io-Tahoe. Hitachi Vantara has enabled integration across these offerings within Lumada DataOps.
- **Differentiation in IoT and edge data integration scenarios:** Due to Hitachi's balanced focus on supporting both IT and OT use cases, Hitachi can differentiate in scenarios that require integrating data from IoT devices at the edge and merging that with data from traditional sources such as databases and file systems.
- **Support for augmented data integration in specific areas:** Hitachi provides support for optimizing data persistence across multiple clouds by analyzing factors such as cost, location, governance and latency. Automation is also used for intelligent data tiering on HDFS files, dynamic selection of execution runtimes, and discovering relationships between datasets. Although there's still a lot of room for improvement here, this is a good start.

Cautions

- **Market traction reliant on Pentaho:** The low number of mentions for Hitachi Vantara among Gartner's client interactions and product reviews is indicative of slowing market momentum in data integration use cases. Customers and prospects (on Gartner inquiry calls) generally equate Hitachi Vantara to just Pentaho Data Integration, being unaware that this has been included in Lumada DataOps, providing many additional data integration capabilities, such as data virtualization via Data Services. Enterprises that do recognize capabilities beyond Pentaho believe these to be mainly relevant to IIoT use cases.
- **Nascent support for differentiating components of the data fabric:** Although Hitachi Vantara has a vision for helping customers optimize their data fabric through its data integration tools and data catalogs, support for certain differentiating components of the data fabric, such as semantic modeling and active metadata support, is at a nascent stage. Hitachi is building upon existing capabilities here, such as AI-based semantic tagging of data.
- **Limitations in cloud-native deployments:** Hitachi's data integration capabilities are not yet available as a fully managed iPaaS across public cloud infrastructure providers such as AWS and Azure. Containerization support for Pentaho was added recently, through which deployments on the public cloud of choice are now possible.

IBM is a Leader in this Magic Quadrant; in the previous iteration of this research, it was also a Leader. IBM is headquartered in Armonk, New York. IBM Cloud Pak for Data (which includes DataStage Enterprise Plus Cartridge), IBM Cloud Pak for Integration (for application integration scenarios), Cloud Pak for Data as a Service (which includes DataStage as a Service, Watson Query as a Service and Watson Knowledge Catalog as a Service) and IBM Data Replication, all target a range of data integration use-case scenarios. The vendor's customer base for this product set is more than 10,000 organizations. Its operations are global, and its clients tend to be enterprise B2B and B2C organizations in the banking and financial services, insurance, healthcare and pharmaceuticals industries.

Strengths

- **Support for the data fabric design:** IBM software has collaborated with IBM research to embed capabilities for augmented data integration into its Cloud Pak for Data (CPD) platform and services. The incorporation of capabilities to capture and activate metadata in Watson Knowledge Catalog, the ability to support DataOps patterns for improved orchestration and agility, and the utilization of knowledge graphs to support semantic modeling and taxonomy to ontology mapping for unstructured content have further improved its support for data fabric use cases.
- **Comprehensive portfolio for operational and analytical use-case support:** IBM has a comprehensive tools portfolio within CPD that includes DataStage (for bulk/batch integration), IBM Cloud Pak for Integration (for application integration and API management), Watson Query (for data virtualization), IBM Data Replication (for data replication and synchronization) and IBM Streams (for stream data integration scenarios). Along with these capabilities, IBM CPD is well-integrated with other data management technologies including data quality, MDM and data governance.
- **Modular architecture and DataOps enablement:** IBM's data integration tools are delivered as tightly integrated and yet loosely coupled services on Red Hat OpenShift (a Kubernetes-based platform). Clients praise IBM's remote runtime capabilities, which reduce egress costs by allowing developers to build pipelines once and push down workloads to the execution environments of their choice. IBM's support for CI/CD and integration with Git (for versioning), Jenkins (for task scheduling) and other third-party task and workflow managers is highly rated.

Cautions

- **Data replication product maturity:** IBM delivers change data capture (CDC) technology through currently supported IBM Data Replication packages. However, some of its customers are still on its legacy CDC tooling. Reference customers of IBM have reported a below-par UI for this product set. Some customers have highlighted challenges with workload monitoring, performance optimization and high availability. IBM should ramp up its migration to the IBM Data Replication services on IBM Cloud Pak for Data

(from legacy IBM CDC offerings) and improve its capabilities for stream data integration support to ensure that existing clients don't have to evaluate third-party data replication solutions.

- **Lack of clarity on migration and upgrades:** Although IBM is aggressively trying to migrate legacy IBM Infosphere Information Server customers to IBM Cloud Pak for Data, some customers report lack of clarity around best practices and license portability, and cite the lack of a well-structured migration and upgrade roadmap. Although IBM does provide embedded migration tooling to help assist with migration to CPD, customers have little knowledge of this capability.
- **Market perception and revenue growth deceleration:** Some IBM prospects (on Gartner's client inquiry service) continue to have a perception that IBM's tools are expensive, complex to implement and targeted toward large organizations that possess strong data engineering skill sets. Gartner's analysis of our proposal review data suggests that IBM's tools were rarely evaluated or considered by SMEs unless they were existing IBM customers. IBM's positioning of Cloud Pak for Data as a modular set of services (rather than an end-to-end platform), and pivoting to a serverless, metered licensing model, should alleviate some of these concerns.

Informatica

Informatica is a Leader in this Magic Quadrant. In the previous iteration of this research, it was also a Leader. Informatica is headquartered in Redwood City, California. It offers several on-premises data integration products including PowerCenter, PowerExchange, Data Engineering Integration, Enterprise Data Preparation and Data Engineering Streaming. It also provides various data integration services as part of its Intelligent Data Management Cloud (IDMC) platform including Cloud Data Integration, Cloud Data Integration Elastic, Cloud Mass Ingestion, Cloud Integration Hub and Cloud B2B Gateway. Informatica has more than 5,700 customers for these product lines. Its operations are geographically diversified, and its clients are distributed across multiple industries, with the top being in the financial services, healthcare and public sectors.

Strengths

- **Augmented data integration support:** One of the main reasons highlighted by customers to evaluate and select Informatica's data integration tools is its automation support for complex and repetitive data integration tasks. Informatica has significantly invested in CLAIRE, its active metadata-driven ML engine, which performs continuous analysis over all collected metadata to significantly automate schema drift, data pipeline orchestration, performance monitoring, and optimization and data modeling.
- **Improved pricing and licensing aligned to cloud adoption:** Informatica has pivoted to a less-complex, consumption-driven licensing model based on Informatica Processing Units (IPUs). This common unit of capacity can be used across its entire set of cloud

services offered under the IDMC umbrella. Users can subscribe to a certain number of IPUs (based on forecast usage), which can then be used interchangeably across all its major data integration product lines. Early adopters of this new pricing model report ease of understanding, adoption and scaling.

- **Operational data integration use-case delivery:** Customers looking for a technical implementation of the data hub architecture to support operational data integration use cases report Informatica's solutions as mature. The Cloud Integration Hub offering is frequently evaluated and selected by customers for its ability to support all data modalities (including batch, virtual, streaming and API-based integration) and for its ability to integrate, govern and share across a multicloud/hybrid environment for data and application integration scenarios.

Cautions

- **Reported challenges with PowerCenter to Informatica Cloud migration:** Some customers report challenges while migrating from PowerCenter to Informatica Cloud. Informatica provides a migration utility tool (that automated a certain percentage of manual mapping conversion tasks) at an additional cost. Some customers reported that their developers had to invest time so that they could efficiently utilize the upgrades (and certain new features) of Informatica Cloud to optimize existing integration workloads in the new cloud environment. Some early migration customers reported the need for manual workarounds and postmigration testing. Informatica has made investments in this area by training and certifying various global system integration partners including Accenture, KPMG and Deloitte that can assist Informatica customers in their migration projects.
- **Go-to-market overly focused on end-to-end data management scenarios:** Informatica delivers a broad portfolio of tools to support a broad range of data integration scenarios. Although this is well-appreciated by large enterprise clients, SMEs and business departments looking to start their journeys with basic data ingestion pipelines often find Informatica's product set and messaging (focused as it is on augmented integration, CLAIRE and data fabric) as overwhelming and complex to navigate. Informatica should balance end-to-end best-of-breed functionality with targeted best-fit engineering tools to support "land and expand" strategies. To address some of these challenges, Informatica recently launched a free Data Loader service to support simplified workflows for building data pipelines.
- **Call for DataOps-related enhancements:** Some customers stated that they were unaware of how Informatica's data integration tools integrate and interoperate with popular third-party or open-source orchestration and task workflow management tools like dbt, Apache Airflow, Luigi, Prefect and Dagster. Data engineers appreciate the low-code integration support of Informatica, but stated that they were not aware of its extensibility features to high-code repositories for certain use cases that require coding. Some

customers also called out the need for improvements in change management, versioning and CI/CD capabilities within Informatica's tool portfolio.

K2View

K2View is a new entry to this Magic Quadrant and is positioned as a Niche Player. It is headquartered in Yokneam, Israel, and offers the K2View Data Product Platform as its data integration tool. The customer base includes more than 50 organizations, mainly in North America and Europe. Clients are commonly in large enterprises in the communications and media, financial services and healthcare verticals.

Strengths

- **Business focus and usability:** K2View is clearly focused on addressing business challenges through its data integration technology. As opposed to other products, which may be targeted toward technical personas, K2View's strategy and go-to-market aligns with common business outcomes and use cases, such Customer 360, legacy data migration, test data management, and operational intelligence for real-time insights (such as churn prediction, credit scoring or fraud detection).
- **Unique "Micro-Database" architecture:** K2View supports a distributed and scalable architecture where the dataset for each business entity is managed in its own Micro-Database. This allows K2View to concurrently manage billions of Micro-Databases with near-real-time data movement, including the ability to apply in-flight data transformations.
- **Flexible data virtualization capabilities:** K2View offers a capability called "dynamic data virtualization," in which some data can be accessed from the source via virtualization and other data can be accessed from a Micro-Database, which is asynchronously synced with the sources. This ability to decide which data will be stored physically, and which will be virtualized, provides advantages in terms of performance optimization and load management on source systems.

Cautions

- **Low market visibility and mind share:** K2View is not as well-known as some incumbent vendors in this market. While not necessarily a drawback, some Gartner clients evaluating data integration tool vendors were slightly hesitant to evaluate K2View due to its relatively low market visibility and mind share. K2View was mentioned infrequently by Gartner clients (on our client inquiry service) especially when those clients were evaluating data integration tools for analytical use cases.

- **Augmented capabilities are limited:** K2View has some augmented capabilities, but these are still developing. For example, K2View is not currently able to use AI/ML algorithms to automate or augment sophisticated data transformation, and it cannot adapt existing data transformations in a proactive, nonhuman-initiated fashion.
- **Pricing is not fully transparent:** K2View uses multiple pricing models, including traditional pricing based on the number of data sources, contract length and level of support. The company also has a consumption-based subscription pricing model, which is based on data reads/writes and data storage. Although these pricing models are also used by other vendors, a lack of detail on the vendor's website means that it is not easy for potential customers to evaluate the pricing structure at a glance and forecast ongoing spend and TCO with K2View as they scale their usage.

Matillion

Matillion is a Niche Player in this Magic Quadrant, the same as last year. It has dual headquarters in Denver, Colorado, and Manchester, U.K., and offers Matillion ETL and Matillion Data Loader as its data integration tools. The customer base includes more than 1,150 organizations, mainly in North America and Europe. Clients are commonly in the business services, software and manufacturing sectors.

Strengths

- **Focus on cloud data engineering:** Matillion offers two products: Matillion ETL is used for building data pipelines, transforming data and preparing data for consumption by analytics tools. Matillion Data Loader focuses on extracting data from source systems and loading it into cloud data platforms. Matillion uses an ELT approach, leveraging SQL-based push down transformations and native optimizations in cloud data platforms.
- **Strong market momentum, recognition and third-party partnerships:** Matillion has deep partnerships with popular data management vendors and system integrators, and was named Snowflake Technology Partner of the Year for Data Integration in 2021. Matillion has a strong integration with Snowflake for out-of-the-box modeling into Snowflake's data warehouse environment. Matillion also partners with Collibra for data governance, and it conducts co-selling and provides integrations with multiple other data and analytics companies, such as Databricks and ThoughtSpot.
- **Deployment options address common data sovereignty issues:** Matillion is deployed as a virtual machine image in the customer's cloud data warehouse, so the client company has full control over the data, which never leaves the customer's environment. This

allows Matillion to address data sovereignty issues and privacy concerns, and means that the tool can be used by government agencies, such as in the AWS GovCloud environment.

Cautions

- **Limited support for metadata and data governance:** Matillion does not have full support for all metadata management capabilities, such as semantic modeling, taxonomy and ontology mapping, knowledge graphs, impact analysis, and active metadata support for automation. As another example, Matillion partners with other metadata management and active metadata vendors, such as Collibra, for data lineage, and does not currently offer these features natively.
- **Limited data observability and data virtualization support:** Matillion currently does not offer mature capabilities related to data observability, such as data pipeline monitoring and ML-based recommendations for optimizing data integration workload performance. These features are on the company's product development roadmap. For data virtualization, Matillion supports basic virtual data access via external tables, but lacks advanced data virtualization capabilities. For example, support for external query acceleration engines like Impala or Presto and dynamic query optimization is currently lacking.
- **Customers report difficulty with upgrades:** Although Matillion does a good job of introducing new features, and has regular product releases, some customers feel that the company's testing of new features and communication about product changes could be improved.

Microsoft

Microsoft is a Leader in this Magic Quadrant. In the previous iteration of this research, it was also a Leader. It is headquartered in Redmond, Washington. It offers SQL Server Integration Services (SSIS) for on-premises data integration tasks, Azure Data Factory (ADF) for hybrid data integration tasks, and Power Query for data preparation tasks. Its operations are geographically diverse, and its clients range from small and midsize businesses to large enterprises. Gartner estimates a customer base of about 20,000 companies.

Strengths

- **Native integration reduces complexity and improves interoperability within the Azure data ecosystem:** ADF's native integration with other Azure cloud services (like ADLS, Microsoft Purview and Microsoft Synapse Analytics) makes it the most suitable data integration tool choice when selecting an Azure-native data ecosystem. For example, lineage metadata from ADF can be accessed within Purview along with business metadata.

- **Product strategy focused on both enterprise and departmental use cases:** Microsoft positions ADF and SSIS for technical roles with its data engineering capabilities, which include curating trusted data assets and managing the full life cycle of data pipelines. It positions Power Query for citizen roles with its core data preparation capabilities and ease of use to simplify data access.
- **Go-to-market partnerships:** Microsoft leverages several partnership channels with independent software vendors (ISVs) operating on Azure cloud (such as Snowflake and Databricks for dbPaaS), and also with global system integrators (SIs). It offers several jump-start data integration projects with demo code libraries and tutorials. It also operates a lift-and-shift migration program of SSIS workloads to ADF toward retaining legacy customers during their data modernization efforts.

Cautions

- **Integration complexities in hybrid architectures:** Microsoft's portfolio of cloud and on-premises data integration tools is good enough to support various deployment options such as public cloud, private cloud and on-premises. However, it leaves users to address the decisions around data placement (where they will manage specific datasets) and workload placement (where they will run diverse data processing workloads).
- **Cloud workload performance and spend prediction:** Although the pricing model of Azure's data integration tooling is transparent and flexible (supported by the Azure price calculator and cost management tooling), some customers that are not well-acquainted with the Azure ecosystem can find cost planning difficult when determining scenarios and budget projections. Some report the need to add multiple nodes to speed up data processing, which is the least economical route.
- **Error reporting and debugging is sometimes inconsistent:** Several Gartner Peer Insights reviews highlight the lack of clarity on error messages and how troubleshooting becomes a challenging task. In addition to product improvements, Microsoft should also provide DataOps best practice guidance such as code templates and libraries, video tutorials and partner accelerators library.

Oracle

Oracle is a Leader in this Magic Quadrant, the same as last year. Based in Austin, Texas, Oracle offers the Oracle GoldenGate (GG) platform, Oracle Data Integration Suite (ODI), Oracle Big Data SQL (BDSQL), data integration services within Oracle Integration Cloud and Oracle Cloud Infrastructure (OCI) Data Integration Services. Oracle's customer base for these products is more than 16,000 organizations. Its operations are geographically diversified, and its clients are primarily in the financial services, telecommunications and retail/e-commerce sectors.

Strengths

- **Technical superiority in data replication:** Oracle GoldenGate stands out in the data replication/change data capture space through its multithreaded log capture API, edge deployments and microservices-based architecture. GoldenGate is often deployed to support mission-critical applications that need to always be highly available.
- **Balanced focus across OCI and multicloud deployments:** Although OCI Data Integration is a strong part of the OCI ecosystem, Oracle continues to deliver enhancements for its stand-alone data integration products for multicloud deployments. Native integration for Snowflake with Oracle GoldenGate (with an option for external staging in other public cloud object stores) is an example of this multicloud focus.
- **Clear vision for integration convergence:** Oracle's vision of combining data integration with application integration by designing reference architectures utilizing Oracle data integration products alongside Oracle Integration Cloud sets it up for long-term differentiation.

Cautions

- **Relatively limited adoption for data virtualization:** Oracle isn't evaluated for data virtualization as frequently as other stand-alone data virtualization product vendors. Due to relatively limited adoption for Big Data SQL as compared to other Oracle products for data integration, some customers find the current user community around the product to be too small. Oracle's recent push to enable virtual data access across multiple clouds should accelerate adoption going forward.
- **Perception of alignment to complex use cases only:** Due to Oracle's track record of supporting highly complex transactional workloads, customers often do not consider Oracle for integration scenarios requiring a "good enough" solution, such as low-code data ingestion into cloud data warehouses. As of now, end-user organizations remain largely unaware of the low-code capability and augmented user experiences within Oracle's products that could support Oracle's penetration into the SMB market.
- **Improvements required in troubleshooting and monitoring:** Some customers have expressed dissatisfaction with the level of in-product assistance provided for troubleshooting errors. Some have also asked for improvements in monitoring capabilities to detect errors more quickly.

Palantir

Palantir is a new entrant to this Magic Quadrant and is positioned as a Visionary. It is headquartered in Denver, Colorado and offers the Palantir Foundry Data Integration Suite as its data integration platform. The customer base includes 237 organizations, mainly in North America and Europe. Clients are commonly in the following sectors: government, healthcare and pharmaceuticals, law enforcement, aerospace, and oil and gas.

Strengths

- **Highly skilled embedded engineers that can build and maintain complex data applications:** Palantir's clients are mainly large organizations that use the company's forward deployed engineers to build, deploy and maintain end-to-end analytic applications. Reference customers praise the deep support provided when building ontologically aligned and taxonomically defined data services, which often require complex data transformation, data orchestration and semantic modeling.
- **Data fabric vision:** One of Foundry's biggest differentiators is its integration of a fully featured semantic layer – the Foundry Ontology – within the same stack of capabilities as connectivity, data transformation, metadata management and data security. Integrated data can be mapped to a graph-oriented semantic layer, which can synthesize multimodal attributes. Foundry Ontology, once developed, supports the representation of future integrated data as knowledge graphs. Additionally, the Foundry semantic engine can extend the Foundry Ontology with its Ontology Gateway – an API layer supporting access to all elements in ontologies and associated pipelines.
- **Extensive support for data security, privacy and governance:** Palantir's clientele includes some of the world's largest defense agencies, state and central governments, and law enforcement agencies. The platform preserves the security states (including role-based, classification-based, and purpose-based models) across all data and model versions. It enables branching of all data and code in a secure sandbox; provides security-aware search, indexing, and cataloging; and extends through security APIs that can be used to integrate with existing workflows and third-party security systems.

Cautions

- **Lack of knowledge transfer and skills availability:** Palantir customers in Gartner client interactions emphasize reliance on forward-deployed engineers to maintain their data applications. Although most customers are satisfied with the quality of this arrangement, they state challenges with knowledge transfer that would reduce their reliance on Palantir support and specify that sourcing Palantir skills from the market has been challenging. Palantir's incorporation of a knowledge transfer program and a first wave of joint forays into the market with consulting partners should alleviate some of these concerns.

- **High-cost, high-value perception:** Palantir targets large organizations that require support for complex data management and analytics use cases. Also, Palantir's pricing model – in which it charges organizations on "configuration of and access to a scoped use case" – reflects the development of end-to-end data integration use cases. As a result, small to midsize organizations have a perception of Palantir as being too comprehensive (and expensive) for their basic data integration requirements. Palantir must demonstrate a more modular approach to better position itself for SMEs and midsize organizations.
- **Extensibility and integration with other data management ISVs:** Some Palantir customers report that the vendor has a tendency to engage in scope expansion that targets the most difficult integration use cases by including mundane data integration tasks that could be managed with lower-cost tools. Even though Palantir has recently entered into partnerships with IBM, Microsoft and Google (among others), it must expand on its integration with other incumbent data management vendors to showcase better interoperability with the organizations' existing data management infrastructure.

Precisely

Precisely is a Challenger in this Magic Quadrant, the same as last year. Based in Burlington, Massachusetts, Precisely offers Connect, Ironstream and Spectrum as its data integration products. The vendor's customer base for this product set is more than 2,500 organizations. Its operations are geographically diversified, and its clients are primarily in the financial services, insurance and healthcare sectors.

Strengths

- **Differentiating capabilities for customers' data modernization initiatives:** Many customers use Precisely for moving data from legacy systems (such as IBM iSeries, IBM zSeries and Hadoop-based data lakes) into cloud data warehouses, lakehouses and message queues as part of their data modernization initiatives. Additionally, Precisely's ability to observe, mask and format different mainframe data types is sought after by its customer base, as most other vendors no longer focus on mainframes.
- **Ease of configuration and development:** Precisely's intuitive GUI, which facilitates easy and quick development, configuration and bug fixing, is highlighted as praiseworthy by customers.
- **Strong vision for enabling an open data ecosystem:** Precisely provides products for data integration, data quality and enrichment, master data management, and data governance, as part of its Data Integrity Suite. This is a modular data management platform that can run on open frameworks like Spark and interoperate with third-party data management solutions such as CI/CD tools and data lineage solutions.

Cautions

- **Deployment options are limited:** Precisely is neither available as an iPaaS for cloud-native deployments nor deployable on edge devices to support a distributed data and analytics architecture. Deployment on cloud infrastructure (and the availability of Connect ETL's batch capabilities) as Docker containers mitigates this challenge to a degree. Also, a SaaS-based data integration module within the Data Integrity Suite is on the roadmap.
- **Diminished value for specific data delivery styles:** Although Precisely differentiates itself with a comprehensive data management portfolio, its value diminishes when customers are looking for leading products for specific data delivery styles. For instance, automatic schema drift handling and agent-based architecture for data replication are roadmap items. Precisely is evaluated a lot more frequently when the source for replication is a mainframe system than when it's not. Precisely Spectrum Quality is rarely evaluated as a stand-alone data virtualization product.
- **Lack of alignment to the data fabric vision:** Precisely's limited augmented data integration and active metadata management capabilities continue to hamper its support for data fabric use cases.

Qlik

Qlik is a Challenger in this Magic Quadrant, the same as last year. Based in King of Prussia, Pennsylvania, Qlik targets a range of data integration use cases through its Qlik Replicate, Qlik Compose, Qlik Enterprise Manager, Qlik Cloud Services, Qlik Application Automation and Qlik Catalog products. Its customer base for this product set is more than 3,000 organizations globally. Qlik's operations are predominantly based in North America and EMEA, and its clients tend to be enterprises in the healthcare, financial services, retail and manufacturing sectors.

Note: In October 2020, Qlik acquired Blendr.io, a provider of iPaaS technology for data and application integration. In August 2021, it also acquired Nodegraph, a metadata management startup focused on interactive data lineage, impact analysis and data governance.

Strengths

- **Data engineering use cases for data warehouse automation:** Reference customers of Qlik praise its real-time data replication and complex data transformation in support of data warehouse automation and data lake enablement scenarios. Native connections to legacy databases, file systems, object stores and Kafka (for stream data integration scenarios) were highly rated by Qlik's clients. The maturity of Qlik Compose for rapid development of data models (through automated modeling, DDL and ETL generations) was also cited as a key factor by Qlik's clients for evaluating and selecting Qlik.

- **Key acquisitions improve key capabilities:** Qlik's recent acquisition of Nodegraph has significantly improved the data lineage capabilities of its data catalog service. The acquisition of Blendr.io (which was renamed Qlik Application Automation) extends its application integration, reverse-ETL and API integration. Finally, the acquisition of Big Squid (now called Qlik AutoML), has enhanced its augmented data integration capabilities. Qlik AutoML also augments data preparation tasks within Qlik Data Catalog and provides self-healing schema evolution within Qlik Compose.
- **Self-service data integration support:** Qlik's tooling supports data integration specialists, as well as less-technical personnel in order to make data available through real-time data streaming. The role-based interface of Qlik Compose allows citizen integrators to develop data marts in their data warehouses. On the other hand, the design and administrative tools within Qlik Compose are used by enterprise architects and data engineers when more-robust data models are required.

Cautions

- **Lack of focus on operational data integration:** Although Qlik's focus on data warehouse automation and data engineering continues to gather momentum, its support for operational data integration use cases needs improvement. Several Gartner clients have stated that Qlik should focus on enabling data-hub-based architectures for efficient integration, governance and sharing for complex application integration scenarios.
- **Challenges with performance optimization and synchronization of replication jobs:** Although Qlik Replicate's heterogeneous compare and repair has improved, some customers continued to cite challenges with setting up and configuring replication jobs. They have stated that the setup is cumbersome and that it took them significant time, effort and intervention from Qlik support agents to identify, fix and resynchronize replication workloads, which lead to downtime. Qlik should improve its monitoring, task observability and performance tuning.
- **Limited support for data virtualization use cases:** Qlik's ability to support data federation and virtualization is currently limited to registering datasets in Qlik Catalog and delivering derived datasets via APIs solely for analytics consumption. Customers looking for advanced data virtualization – including creating virtual data hubs and marts, dynamic query optimization, support for external analytics query acceleration tools and MPP engines – will need to evaluate third-party data virtualization tools.

Safe Software

Safe Software is a Niche Player in this Magic Quadrant, the same as last year. It is headquartered in Surrey, British Columbia, and offers the FME Enterprise Integration Platform, which is composed of FME Desktop, FME Server, FME Cloud and FME Mobile. The customer base includes more than 9,300 organizations, mainly in Europe and North America. Clients are commonly in the government, utilities, energy, architectural engineering and construction, and telecommunications sectors.

Strengths

- **Spatial data integration:** Safe Software provides strong support for spatial data integration, spatial data formats, spatial data transformations and processing data from IoT “things” such as internet-connected temperature sensors. Although spatial data is clearly Safe Software’s specialty, the company’s support for nonspatial data types and use cases is also robust. As such, this platform will work for a variety of organizations and data integration scenarios. Safe Software supports geospatial data integration with a range of sources and targets, including DbPaaS vendors such as Snowflake and Amazon Redshift.
- **Execution and support:** Safe Software has clearly packaged deployment options and pricing/licensing models that are targeted toward certain industry verticals. As an example, Safe Software has offerings that are specifically for local governments or utility companies, with pricing based on the population served. Deployment options include on-premises, in the customer’s cloud, and as a vendor-managed cloud offering.
- **Support for nontechnical users via unique, user-centric interfaces:** Safe Software supports business-level users by provisioning a low-code/no-code graphical interface that is part of the data integration platform, but the company is innovating by also offering mobile apps and augmented reality. FME AR allows users to interact with data via their smartphones, such as visualizing water pipe locations underground while the user walks along the street. These innovative features can help nontechnical users to leverage data in their day-to-day work environments.

Cautions

- **Limited traction outside of spatial data integration:** Although Safe Software’s platform is capable of handling both spatial and nonspatial data, customers without spatial data use cases are less likely to choose Safe Software. This is driven by the company’s product marketing — it is clearly pitched as a spatial data integration solution — which limits the number of customers that consider purchasing this product.
- **Metadata capabilities are still developing:** FME includes a “SchemaScanner” transformer to harvest the schema from the source data, which can then be compared to what users expect, but this process is not fully automated. The current workflow is a step in

the right direction toward using metadata to detect patterns and make recommendations, but the company's support for metadata activation is not yet fully developed.

- **Limited augmented capabilities:** The product's breadth of augmented capabilities is limited. This includes a lack of metadata support as mentioned above, but also the lack of concepts such as knowledge graph support, FinOps and out-of-the-box techniques to augment or automate sophisticated transformation logic (e.g., using ML models).

SAP

SAP is a Leader in this Magic Quadrant, the same as last year. It is based in Walldorf, Germany. It offers the following data integration tools: SAP Data Intelligence, SAP Data Services, SAP Landscape Transformation (SLT) Replication Server and SAP Integration Suite. It also offers integration capabilities within the SAP HANA platform. Its operations are geographically diverse, with a customer base of more than 18,000 companies. The top-three industries it supports are automotive, consumer products and public sector.

Strengths

- **Native integration with SAP applications:** SAP data integration solutions are tightly integrated with various SAP applications like ECC, B/W and HANA. Native integration reduces complexity and improves interoperability within the data ecosystem. In addition, its data integration solutions are preintegrated with data quality, data governance, data preparation and data stewardship capabilities.
- **Product strategy focused on unification, hybrid and scale:** SAP leverages its flagship on-premises offerings (SAP Data Services and SLT Replication Server) into a hybrid data management strategy (with SAP Data Intelligence Cloud) to meet the needs of all customers. It delivers an end-to-end integration strategy for its customers by enabling them to combine and switch between multiple integration styles across on-premises and cloud. SAP Integration Suite offers a unified integration platform for end-to-end process and data integration, as well as API management, among other capabilities.
- **Centralizes metadata in its data catalog:** SAP Data Intelligence offers a rich set of metadata capabilities such as cataloging, search, lineage, data quality checks, PII detection and data access logging for auditability. In addition, its active metadata capabilities enable data and metadata discovery to drive recommendation engines, warnings, rules, and semantic discovery, including a business glossary in the data catalog.

Cautions

- **Perception of high license costs:** Several SAP data integration customers report high licensing costs. Also, starter and small organizations rarely consider SAP data integration products due to its high price perception. However, SAP's cloud pricing strategy provides specific customer solutions, such as bundling multiple products into single licenses and sharing resources, which might reduce the overall cost.
- **New customers struggle with initial setup:** Several new SAP data integration customers known to Gartner have expressed concerns around initial setup complexities and delays. Adding to this, there is a shortage of skilled expertise in the market for its newer products like SAP Data Intelligence. Many customers are depending on SAP service partners to address this challenge. However, SAP is moving toward containerized deployments, which should reduce additional consulting services required for the initial setup.
- **Limited integration with non-SAP products:** A significant number of SAP customers report challenges with the lack of functionality of SAP data integration tools to replicate data from SAP tables to non-SAP targets (like AWS, GCP or Azure). Customers report that SAP's data integration tools (like SAP Data Services and SLT Replication Server) are not suited for these tasks (and sometimes have licensing concerns for replicating data to non-SAP targets). Therefore, they procure third-party tools like Theobald, Aecorsoft and SNP Glue that are able to query the SAP application layer and replicate data to non-SAP targets. SAP Data Intelligence Cloud is expected to address this gap.

SAS

SAS is a Challenger in this Magic Quadrant, the same as last year. Based in Cary, North Carolina, SAS offers the following data integration products: SAS Studio Engineer on SAS Viya, SAS Event Stream Processing on SAS Viya, SAS Intelligent Decisioning, SAS In-Database Technologies, SAS Data Management and SAS Data Integration Server. The vendor's customer base for these products is around 16,000 organizations. Its operations are geographically diversified, and its clients span many industries with the top sectors being financial services and government.

Strengths

- **Flexibility of design options for data engineers:** SAS Studio Engineer on SAS Viya enables data integration jobs to be designed either in a GUI; by writing custom code in SAS, SQL and Python; or by invoking code within databases such as Hadoop, Databricks, Azure Synapse or Teradata using SAS in-database technologies. Once designed, these jobs can run in batch mode, streaming mode or "as a service."

- **Continued support for collaborative data engineering:** SAS Studio Engineer provides persona-based interfaces, including specialized interfaces for both nontechnical business users and analysts. SAS Intelligent Decisioning is a business-user-focused tool to develop business rules to monitor and validate data. SAS's AI-driven data transformation suggestions aid in accelerating and streamlining data preparation processes.
- **Enhanced integrations for DataOps support:** SAS integrates with Git, Bitbucket, Jenkins and Azure DevOps among others for building DataOps pipelines. SAS artifacts can work with CI/CD pipelines using APIs and CLIs. In addition to SAS's native scheduler, SAS Job Flow Scheduler, integration with third-party process orchestration solutions like Apache Airflow, Dagster and Prefect is provided.

Cautions

- **Challenges with SAS's vision as an independent data integration vendor:** Customers tend to evaluate SAS for its data integration capabilities primarily when they've already invested in its analytics and data science solutions. SAS is playing into this tendency by positioning SAS Studio Engineer as a part of SAS Viya, which is a unified AI, analytics and data management platform. This further exacerbates the challenge of positioning itself as an independent data integration vendor.
- **Issues with pricing:** Customers point out high license costs as one of the main areas of concern for SAS Data Management. With SAS Viya being a cloud-native platform, customers expect SAS Studio Engineer to be available at more competitive price points and to complement its authorized user-based model by embracing modern consumption-based models.
- **Lacks vision for data fabric support:** SAS does not provide mature capabilities to support active metadata management or semantic data modeling. Strategic partnerships with vendors across technologies like metadata management and data governance are few, thereby limiting opportunities to co-sell unified solutions for complete data fabric support. Partnerships with vendors through the Egeria project are a step in the right direction.

SnapLogic

SnapLogic is a Visionary in this Magic Quadrant; in the previous iteration of this research, it was also a Visionary. Headquartered in San Mateo, California, SnapLogic offers the SnapLogic Intelligent Integration Platform as its data integration offering. It operates mostly in North America and EMEA, with a customer base estimated at 1,400 organizations from diverse sectors including technology, consumer goods, retail and manufacturing.

Strengths

- **Product strategy focused on prebuilt connectors called Snaps:** SnapLogic offers more than 600 dedicated Snaps that enable code-free integrations such as flattening nested data structures, bulk load and streaming data. Snaps help developers to connect multiple endpoints together, eliminate data silos, obtain complete insights, and yield high-performing business results.
- **Productivity driven by augmented guidance:** SnapLogic empowers less-technical citizen roles to reduce their reliance on highly skilled data engineers through its ease of use and embedded ML guidance, called "Iris." A single platform with multiple interfaces for all personas makes it more compelling for both data preparation and data engineering use cases.
- **Established role-based go-to-market strategy:** SnapLogic facilitates quicker adoption and onboarding of prospects through its simple pricing and trial versions. It targets departmental leaders with its Fast Data Loader managed service, which can load data into cloud data stores with prepackaged, ready-to-run pipelines.

Cautions

- **Market perception of cloud-only relevance:** SnapLogic suffers the market perception that its integration capabilities are limited to cloud applications despite it offering on-premises solutions. This adversely affects some competitive evaluations known to Gartner. SnapLogic is educating its prospects and customers for its broader reach in the data integration tools market.
- **Limited guidance and support for implementations:** Many SnapLogic customers report difficulties around troubleshooting issues, and noted poor product documentation and training manuals. Deployments in complex scenarios have raised customer expectations about SnapLogic's implementation support and guidance.
- **Limited governance support:** SnapLogic lacks some capabilities such as scoring/certifying data assets for ownership and tagging/masking sensitive data in support of data governance use cases. Some customers state that they need comprehensive data quality assistance.

Software AG (StreamSets)

Software AG (StreamSets) is a new entrant to this Magic Quadrant and is positioned as a Niche Player. It is headquartered in San Francisco, California, and offers the StreamSets DataOps Platform as its data integration tool. The customer base includes more than 150 organizations, mainly in North America and Europe. Clients are commonly in the financial services, technology, telecommunications, insurance and healthcare sectors.

Note: On 19 April 2022, StreamSets was acquired by Software AG, a provider of API management, iPaaS and business process management software.

Strengths

- **Support for streaming, message queues and event-broker platforms:** StreamSets takes a streaming-first approach to ingestion, enabling continuous data delivery. Supported streaming and event brokers include: Apache Kafka, Amazon Kinesis, Amazon SQS, Azure Event Hub, Azure IoT Hub, Google Pub/Sub and RabbitMQ. Because many vendors don't have mature stand-alone DBMS independent stream data integration and data replication, this is a differentiator; StreamSets displays particular strengths in these areas.
- **Engineering-focused DataOps, observability and orchestration:** The StreamSets platform supports orchestration for combining multiple data delivery styles, such as streaming ingestion into a data lake, followed by batch ETL from a data lake to a data warehouse. StreamSets has mature automatic schema drift detection and resolution, and can send alerts to users. The platform has built-in version control and also supports third-party CI/CD tools like GitLab and Jenkins. This combination of capabilities yields the kind of stable data integration pipeline delivery that appeals to data engineers for analytics and data science use cases.
- **Future-oriented market understanding and product alignment:** StreamSets has a strong vision for the future of data integration, especially as it relates to augmentation, automation and monitoring. As an example, StreamSets has a single-pane-of-glass control plane for management and monitoring, which provides real-time operational transparency. Features like this improve operational intelligence and pave a clear roadmap for future enhancements in augmented data integration and FinOps support.

Cautions

- **Postacquisition uncertainty:** StreamSets' recent acquisition by Software AG has raised some postacquisition concerns related to its product roadmap, packaging, positioning, licensing and postimplementation maintenance and support. As a parent company, Software AG is more relevant in the application integration space, with the acquisition of StreamSets giving it an entry into the data integration market. Software AG has committed to its investments in StreamSets as a stand-alone product along with a detailed roadmap regarding upcoming enhancements, which should alleviate some of these concerns.
- **Limited native support for data cataloging and data lineage:** StreamSets does not currently support data cataloging or lineage from within the DataOps platform, but does integrate with third-party tools like Collibra and Apache Atlas for metadata management support.

- **Market traction currently limited to stream data integration:** StreamSets does not have strong name recognition beyond stream data integration use cases. Although StreamSets has invested effort into its CDC capabilities, organizations who are looking for other data delivery styles, such as bulk/batch data movement or data virtualization, are more likely to consider other vendors.

Talend

Talend is a Leader in this Magic Quadrant; in the previous iteration of this research, it was also a Leader. Talend is headquartered in San Mateo, California. Its data integration tools include the Talend Data Fabric (which includes the Talend Studio, Stitch, Talend Big Data, Management Console, API Services, Data Inventory, Pipeline Designer, Data Preparation and Data Stewardship) and the Talend Data Catalog. Talend has more than 7,200 licensed customers for this product line. Its operations are geographically diversified, and its clients represent companies in a variety of sectors, such as media and services, financial services and manufacturing.

Note: Thoma Bravo, a private equity investment firm, announced the acquisition of Talend on 2 September 2021. Talend acquired Gamma Soft, a provider of log-based change data capture technology for data replication and synchronization support, in April 2022.

Strengths

- **Focus on improved data quality while creating data pipelines:** The Talend Data Fabric platform leverages metadata analysis to provide an initial “trust score” to data pipeline designers. This is an aggregate view of the quality of metadata that measures the validity, completeness, discoverability and usage/popularity of the discovered datasets in the Talend Data Inventory Module. This can be utilized by data engineers to build more optimal data pipelines in the Talend Studio and Pipeline Designer, and to reduce data quality and technical debt caused by schema drift.
- **Improved capabilities for self-service data preparation and data engineering:** Talend’s reference customers praise its governed approach to self-service data preparation. Power users can inventory distributed data assets in the Talend Data Catalog. They can then view and improve the quality of the data in the Talend Data Inventory module and allow data stewards to enforce policies in Talend Data Stewardship. Once the stewards are content, the power users can perform their own transformations using the low-code environment of the Talend Pipeline Designer, or invite data engineers to perform complex transformations and operationalize the self-service findings in Talend Studio.
- **Improved support for data replication scenarios:** Talend recently concluded the acquisition of Gamma Soft, a provider of log-based change data capture technology (CDC). This acquisition will enable Talend to augment its existing data replication capabilities

(within the Talend Studio) with the native CDC adapters and APIs from Gamma Soft for real-time interenterprise data sharing and database synchronization scenarios.

Cautions

- **Data observability challenges:** Several Talend customers have called out the need for improved data observability within Stitch and other data integration modules of the Talend Data Fabric. Customers have also requested improved diagnostics, root cause analysis and more-efficient performance monitoring of jobs. Others have asked for deeper integration between the Talend Remote Engines (that allow customers to run jobs in different environments) and the Talend Management console for better tracking and management of the remote agents.
- **Support challenges:** A few Talend customers have reported concerns with after-sales support, training and documentation. Customers stated the need for improved documentation with better examples, more descriptive error messages (so that errors can be more easily replicated for seeking Talend support) and, in some cases, faster response and turnaround time from Talend when the customer is on the basic support level.
- **Concerns with pricing and scaling:** A small but significant number of Talend clients have reported challenges with unpredictable pricing as they scale data volumes with Stitch. Some customers also reported concerns with price increments during license renewals with other Talend data integration tools. Talend should work with prospects to better gauge their requirements and be more proactive in suggesting pricing options that balance a land-and-expand strategy with options to scale. Talend introduced pricing and packaging changes in 2021 to address some of these challenges.

TIBCO Software

TIBCO Software is a Challenger in this Magic Quadrant; in the previous iteration of this research, it was also a Challenger. TIBCO is based in Palo Alto, California. Its data integration tools portfolio includes TIBCO Data Virtualization (TDV), TIBCO Cloud Integration, TIBCO Messaging, TIBCO Streaming and TIBCO OmniGen. The vendor's customer base for this product set is more than 7,500 organizations. Its operations are geographically diversified, and its clients include companies in the financial services, telecommunications and manufacturing sectors.

Strengths

- **Improved interoperability across products:** TIBCO Cloud Metadata is capable of sharing metadata across all of TIBCO's data and analytics products. TIBCO Cloud Passport provides customers the flexibility to utilize various TIBCO Cloud services (including TIBCO Cloud Integration) through a single consumption model. This is part of TIBCO's efforts to position its capabilities as a loosely coupled, highly cohesive offering.
- **Specialization for most data delivery styles:** TIBCO Software has always been rated well in stream data integration through TIBCO Streaming and TIBCO Cloud Events, in messaging through TIBCO Messaging, and in data virtualization through TDV. A recent acquisition extends this portfolio to ETL and ELT as well. TIBCO is now focused on capabilities that can combine these styles, such as streaming data virtualization within TDV.
- **Strong data fabric vision:** TIBCO Agile Data Fabric embeds its AI/ML capabilities into all products through plug-in algorithms for augmented data management. Newly patented technology around data classification and tagging supports creation of business-enriched logical models over data. The user experience across multiple products has also been improved to bolster the data preparation component of the data fabric.

Cautions

- **Lacking in DataOps support:** TIBCO has been relatively slow to react to the rapidly escalating customer requirements around DataOps, such as end-to-end orchestration of data delivery with automated steps for data management code versioning, code testing and code deployment into production. Although TIBCO does support integrations with some CI/CD and version control tools like Git, and TDV Deployment Manager can be used to move models across dev/prod/test environments, a holistic vision for DataOps is missing.
- **Limited traction for the cloud migration use case:** Customers rarely evaluate TIBCO when migrating analytical workloads to public cloud and maintaining a hybrid cloud environment. TIBCO currently provides containerization support for its data integration products, with full SaaS availability already on its 2022 roadmap. Also, although TDV does play a role in this use case, log-based bidirectional data replication is often a requirement here, which is an area of improvement for TIBCO.
- **Need for improved documentation:** Customers have requested more content to be added to the assistance wiki for TDV, better learning materials for users new to TDV and more updated documentation on system integration to third-party platforms. TIBCO is on the path toward addressing this by launching a new digital community for customers and partners that will include customer support and product documentation, among other things.

Vendors Added and Dropped

We review and adjust our inclusion criteria for Magic Quadrants as markets change. As a result of these adjustments, the mix of vendors in any Magic Quadrant may change over time. A vendor's appearance in a Magic Quadrant one year and not the next does not necessarily indicate that we have changed our opinion of that vendor. It may be a reflection of a change in the market and, therefore, changed evaluation criteria, or of a change of focus by that vendor.

Added

The following vendors are included in this Magic Quadrant as they met the inclusion criteria:

- AWS
- Hitachi Vantara
- K2View
- Palantir
- Software AG (StreamSets)

Dropped

- **Adeptia:** Adeptia is dropped from this year's Magic Quadrant because the vendor positions itself as a citizen integration solution focused on self-service data access, integration and sharing. Adeptia did not share the required information to substantiate its ability to meet the latest inclusion criteria for this research (which includes and requires use cases beyond self-service data preparation and integration for citizen integrators).
- **HVR:** Fivetran (already on this Magic Quadrant) acquired HVR in October 2021.

Inclusion and Exclusion Criteria

The inclusion criteria represent the specific attributes that analysts believe are necessary. To qualify for inclusion, the vendor's data integration tool (or tools) must offer at least one "stand-alone" product directly usable by the buyer. The vendors must offer a generally available software product that meets Gartner's definition of a data integration tool. To use the product, customers should be able to

procure the data integration tool as an independent offering and not as a part of some other offerings — such as another form of tool suite, an application or other technology solution — of which the data integration capabilities are an “embedded” subset.

The data integration tool (or tools) must demonstrate various data delivery styles (from the list below) and be flexible enough to combine these styles for delivering various customer use cases:

- **Bulk/batch data movement:** Bulk and/or batch data extraction and delivery approaches (such as support for ETL/ELT) are used to consolidate data from distributed databases and formats. This capability draws on data from across systems and organizational boundaries and can play a role in all use cases in this research.
- **Data virtualization:** Data virtualization executes queries against distributed data sources to create virtual, integrated views of data “in memory.” Virtual views require adapters to data sources, a metadata repository and a distributed query engine that can provide results in various ways for downstream consumption.
- **Data replication:** Data replication implies a simple copy of data and schema from one location to another, always in a physical repository. Replication does not change the form, structure or content of the data it moves.
- **Data synchronization:** Data synchronization focuses on establishing and maintaining consistency between two separate and independently managed create, read, update, delete (CRUD) instances of a shared, logically consistent data model for an operational data consistency use case. Synchronization also maintains and resolves instances of data collision, with the capability to establish embedded decision rules for resolving such collisions (using schema-drift-handling mechanisms and other means).
- **Stream data integration:** This is the ability to address data integration requirements through interoperability with streams/events, including provisioning of data in-stream for enabling downstream consumption, analysis or storage.
- **Data services orchestration:** This covers both message-oriented middleware and API services:
 - This capability allows data integration tools to encapsulate data in messages that various applications can read. Data is exchanged in real time, often via message queues like Apache Kafka or by using message-oriented middleware such as Java Message Service (JMS), IBM MQ and RabbitMQ.
 - This capability allows data as a service, enabled through API design and delivery capabilities, to create and manage outbound API endpoints over existing data assets, and to handle inbound API consumption to ingest internal and external data.

Beyond the data delivery styles called out in the list above, the data integration tools must exhibit the following capabilities to be included in this Magic Quadrant research:

- **Range of connectors/adapters (sources and targets):** This includes native connectors to seamlessly access relational and nonrelational DBMS products, plus access to nonrelational legacy data structures, flat files, XML and message queues, cloud-based data asset types (including data of SaaS applications and cloud data stores), and streaming data. Preference is given to connectors that work out-of-the-box, as opposed to connectors that must be customized by end users.
- **Data movement topology:** Uni-/bi-/multi-directional movement of data across endpoints (e.g., synchronize, compare, broadcast, consolidate) via physical and virtual modes, meeting batch/microbatch/real-time latency requirements.
- **Complex data transformation support:** Capabilities that support complex data processing operations such as fixing outliers, sophisticated parsing (e.g., free-form text mining, telemetry logs, media mining), complex data modeling (e.g., automated data pipeline creation for Data Vault modeling) and creating reusable transformations.
- **Augmented data integration support:** These capabilities improve and optimize data integration operations (such as self-healing schema drifts and autorecovery) using extensive use of metadata (for example, usage data, transaction logs and system workloads) and prepackaged ML algorithms that can inform or automate the tasks to ingest, transform, combine and provision data.
- **Support for data preparation capabilities:** This is the usability of data integration tools both for data engineers and citizen integrators and their suitability to support a range of business roles (e.g., citizen integrators and business analysts) for self-service. The emphasis is on empowering nontechnical staff using various techniques such as low-/no-code data blending and visual exploration.
- **Integration portability:** This is data flow design portability across the infrastructure (at on-premises, SaaS, CSP and virtual private cloud [VPC]), providing workload management capabilities in a clean, safe and portable runtime environment (like containerization).
- **Metadata and data modeling support:** This includes automated metadata discovery (such as profiling new data sources for consistency with existing sources), lineage and impact analysis reporting, and the ability to synchronize metadata across multiple instances of the tool. It also involves an open metadata repository, including mechanisms for bidirectional sharing of metadata with other tools. Capabilities must be provided for extensive use of metadata (e.g., usage data, transaction logs, system workloads) to automate/improve data integration and operations tasks.

- **Data governance and information stewardship support:** Capabilities that assist data governance mandates (e.g., data quality, data lineage) while handling data for meeting specific use cases (e.g., master data management, data sharing). This is the ability to import, export and directly access metadata with/and from data profiling, data quality tools and/or other data-governance-enabling technologies (such as MDM, information stewardship, metadata management and data catalog tooling). Accepting business and data management rule updates from data stewardship workflows and sharing data profiling information with such tools is highly desired. Capabilities that assist data governance mandates (such as data quality and data lineage) while handling data for meeting specific use cases (master data management, data sharing and so on) are also needed.
- **DataOps support:** Change management capabilities to data and related artifacts (e.g., Git integration of data pipelines, data model management), automation (e.g., automated testing), orchestration of data delivery (e.g., CI/CD pipelines) with appropriate levels of security to improve the use and value of data.
- **FinOps support:** Capabilities that enable D&A leaders to iteratively control spending, understand product performance and make choices regarding price-to-performance trade-offs, resulting in optimal allocation of resources in the cloud.
- **Runtime platform support:** This is the ability to deploy and run data integration tools on multiple platforms including Windows, UNIX and/or Linux operating systems.
- **Service enablement support:** This is the ability to deploy functionality as services, including manners in which functionality can be called via a web service interface.
- **Support for the delivery of data integration functionality as cloud services:** This could be done through a hosted, containerized, PaaS, IaaS, or SaaS delivery mechanism. The ability to perform integration across a hybrid and possibly a multicloud and intercloud ecosystem is highly desired. The ability to deliver integration services via a PaaS delivery model is highly desired by customers.

In addition, vendors must satisfy the following quantitative requirements regarding their market penetration and customer base:

- **Revenue or customer count:**
 - Either, generate at least \$30 million in software revenue from data integration tools in the calendar year 2021 – that is, from perpetual license with maintenance, or subscription with support (which would include payment only for data integration

software), or through a consumption-based licensing model where the consumption metrics are being used only for the data integration software (on an annual basis).

- Or, maintain at least **450 subscription or maintenance paying customers** (where “customers” does not mean individual “user” license seats), for its data integration tools in production. (The number of downloads without license or maintenance revenue is informative, but not a qualifying piece of information.)
- **Geography:** Have market presence in **at least three of the following regions** (regional market presence is defined as a minimum of 5% of the revenue of the verified production customer base, as well as the existence of dedicated sales offices or distribution partnerships in a specific region):
 - North America (Canada, Mexico and the U.S.)
 - Central and South America
 - Europe (including Western Europe and Eastern Europe)
 - Middle East and Africa (including North Africa)
 - Asia/Pacific region (Including Japan)
- **Market presence:** Demonstrated market presence will be assessed through internal Gartner search, external search engines, Gartner inquiry interest, technical press presence and activity in user groups. A relative lack of market presence could be determined as a reason to exclude a product/service offering.
- Have a data integration tool service **generally available** as of midnight, U.S. Eastern Daylight Time on 2 April 2022. This includes any new functionality added to the service by the specified date. We do not consider beta, “early access,” “technology preview” or other not generally available functionality or services. Additionally:
 - Any acquired product or service must have been acquired and offered by the acquiring vendor as of 2 April 2022. Acquisitions after this date were considered under their preacquisition identities, if appropriate, and are represented separately until the publication of the following year’s Magic Quadrant.

Vendors that focus on narrow use cases that are too specific for the broader data integration market were excluded. Certain vendor tools were excluded because:

- They focused on only one horizontal data subject area – for example, the integration of customer-identifying data.
- They focused on only a single vertical industry.
- They served only their own internally managed data models and/or architectures (such as providing data integration tools that only ingest data to a single proprietary data repository). Or, they were used by a single data discovery/visualization, analytics/BI tool, data science/ML platform, or DBMS/data management solution for analytics, data lake management, data warehouse automation or cloud ecosystem vendor. These vendors use their data integration tools only to ingest/integrate data into their own repository or within the confinement of their own broader tool/platform or ecosystem.
- They provided data integration as a capability embedded within their broader data management/analytics/data science platform but did not provide a stand-alone/independent or commercially off-the shelf generally available data integration tool product.
- Vendors that only provide support for open-source platforms/frameworks or development platforms, which need to be heavily engineered/customized for specific data integration tasks/use cases and/or are specific to a single data integration/data delivery style (such as stream data integration only).
- Vendors that provide adapters or drivers to various data and analytics sources and targets, thereby indirectly supporting data integration, but these vendors do not market a stand-alone data integration tool.
- Vendors that only provide self-service data preparation tools for citizen integrators, power users, analysts and line-of-business (LOB) users, but these tools do not have the ability to physically move data or operationalize these self-service data flows and models into production through data movement, governance and sharing, if and when needed.

Honorable Mentions

- **Cambridge Semantics** is headquartered in Boston, Massachusetts, and offers Anzo Knowledge Graph Platform. Anzo simplifies the integration, modeling and blending of multirelationship data across silos into insight-rich knowledge graphs at enterprise scale. Users can manage their metadata in the Anzo platform by connecting to data sources and then mapping their relationships. Its ontology management enables users to describe their business concepts, map their data assets, deploy a knowledge graph and

drive business decisions. It also provides deep data integration and unification for AI and ML models. There is a good adoption among enterprises toward data fabric use cases. At this time, we do not see sufficient demonstrable market presence from Cambridge Semantics as a stand-alone data integration product vendor, which is why it was not included on this year's Magic Quadrant.

- **CData Software** is headquartered in Chapel Hill, North Carolina, and offers products under three segments. Under the first segment of data connectivity, it provides CData Drivers and CData Connect Cloud, which have more than 3,000 product SKUs. A product SKU is a unique combination of a specific connector and the technology on which it is supported, such as a CData Salesforce Connector available for JDBC, ODBC, Python, Tableau or PowerBI. CData Connect Cloud provides a library of connectivity via a "Data Connectivity as a Service" cloud offering built for business users. Under the second segment of data integration, it provides CData Sync and CData DBAmp. The former is a log-based change data capture product supporting more than 100 data sources, while the latter is a specialist product supporting Salesforce to SQL Server integration. Under the third segment of application integration, it provides CData Arc, which supports multiple communication protocols for B2B integration for secure Electronic Data Interexchange (EDI), Managed File Transfer (MFT) and APIs. At this time, we do not see sufficient demonstrable market presence from CData Software as a stand-alone data integration product vendor, which is why it was not included on this year's Magic Quadrant.
- **Confluent** is headquartered in Mountain View, California, and offers Confluent Cloud and Confluent Platform as its data integration tools. Confluent Cloud is a fully managed, serverless, and cloud-native service for Apache Kafka. Confluent Platform is a self-managed Kafka distribution. Both products are used for enabling stream data integration and stream analytics, both through Kafka and through a portfolio of tools designed to work with Kafka. Many organizations are using Kafka for streaming, messaging and event-based data integration scenarios. Despite its popularity, organizations struggle to maintain Apache Kafka deployments on their own, which can be operationally burdensome. Kafka does not provide full support for all data integration capabilities (such as orchestration, security and governance) and does not have a schema registry for metadata. Confluent helps address these gaps. Confluent is currently a very popular choice for customers looking to support stream data integration, but was excluded from this year's Magic Quadrant evaluation because it does not provide some essential functions including data virtualization, augmented data integration or data preparation.
- **Data Virtuality** is headquartered in Leipzig, Germany, and offers the Data Virtuality Platform and Pipes as its data integration tools. Data Virtuality uses data virtualization to create a unified semantic layer that helps users access data, even when the data is stored in different locations. The Data Virtuality Platform combines both data virtualization and ETL/ELT, providing flexibility depending on the requirements. Customers connect their data sources to the Data Virtuality Platform, create a data logic to map the connections,

and then deliver data to the data targets. Common use cases include real-time data access, creating a logical data warehouse, data federation, data governance, GDPR compliance and creating a self-service semantic layer. Data Virtuality also provides a simplified version of its platform, which is called “Pipes,” that is a “data loader” tool that is used for data ingestion and replication. Organizations can use Pipes to load data into a cloud data warehouse. Data Virtuality was excluded from this year’s Magic Quadrant because it did not exhibit enough market presence (see our Inclusion Criteria section). Also, Gartner did not see Data Virtuality in a significant number of competitive situations for use-case scenarios not involving data virtualization.

- **DataStreams** is headquartered in South Korea and provides the TeraONE Data Fabric platform, which provides capabilities for data integration, data governance and database management. It also provides stand-alone products for specific data integration patterns, such as TeraStream for batch data processing, TeraStream BASS for streaming ETL, DeltaStream for CDC, TeraONE for Data Lake and TeraONE Super Query for data virtualization. Although data governance is the biggest use case with its IRUDA platform, it supports analytical use cases as well through its own visualization tools, SuperVisual and TeraONE Idea, and through partnerships with Tableau, Qlik and others. At this time, DataStreams does not meet our inclusion criteria for market presence, and therefore was not included in the Magic Quadrant this year.
- **dbtLabs** is headquartered in Philadelphia, Pennsylvania, and offers dbt (data build tool) to transform data within data warehouses. Started as an internal productivity tool at Fishtown Analytics (a service company then), it has become dbtLabs (a product company now). It focuses on simplifying the transformation tasks within ELT use cases. It uses DataOps principles to drive velocity including CI/CD deployment practices and test-driven-development (TDD) frameworks to cut down on unmanaged code. Developers use dbt to express a piece of business logic and manage the end-to-end data pipeline process from development to production. There is a wide adoption of the tool among enterprises and a rapidly growing community of users. However, it did not meet the inclusion criteria on support for core capabilities including data movement topology (does not provide data ingestion) and streaming data integration capabilities.
- **eQ Technologic** is headquartered in Costa Mesa, California, and provides the eQube Data as a Service (eQube-DaaS) platform. eQube-DaaS is closely aligned to the data fabric design and offers a low-code/no-code integration platform that supports multiple data integration techniques (through embedded support for batch, streaming, messaging, API-based delivery, data virtualization and application integration). The eQube-DaaS platform is made up of three loosely coupled offerings: eQube-MI (for data integration and data migration use cases), eQube-AG (for application integration and API gateway) and eQube-TM (for data model management and data transformation maps). eQube’s data virtualization service is integrated as part of the eQube-DaaS platform and plays a critical role in offering data as a service. Asset-heavy industries (including aerospace and defense, auto and machinery, energy, shipbuilding

and high tech) looking to establish a data fabric design for use cases including enterprise search, common data model, API creation, synchronization of multiproduct life cycle management (multi-PLM) systems and their data with popular ERP systems (like SAP) will benefit from eQube's native integration support for industrial applications and IoT systems. Although eQube-DaaS did offer all the data integration capabilities for this market, it did not make the inclusion criteria related to market execution for this year's Magic Quadrant.

- **Google** is headquartered in Mountain View, California, and provides a comprehensive set of data integration capabilities as part of its data management portfolio. Its Google Cloud Platform (GCP) offers Cloud Data Fusion, which is a fully managed and cloud-native data integration service that delivers ETL/ELT capabilities. Additionally, Cloud Data Fusion also supports a low-code data preparation environment for self-service data transformation by citizen integrators. GCP also offers a data replication service, Datastream, that delivers a log-based CDC capability to support real-time data movement to BigQuery (GCPs data warehouse), and other GCP databases including CloudSQL, Spanner, Bigtable and Firestore. For message-oriented data movement, Google offers the Pub/Sub integration service, and for customers looking to support IoT data ingestion and streaming analytics use cases, Google offers the Cloud Dataflow service. GCP has also added an orchestration service called Cloud Composer that supports data orchestration and life cycle management. Finally, GCP offers Dataplex, which centrally manages and governs data across all these GCP offerings. Although GCP offers comprehensive data integration services to support all use cases, it did not meet our inclusion criteria on market presence (specifically due to the low number of Gartner client inquiries mentioning GCP in support of stand-alone data integration use cases).
- **Nexla** is headquartered in San Mateo, California, and offers the Unified Data Operations platform. This platform provides a low-code/-no-code, automation-based approach for diverse roles (such as data analysts, data engineers, business ops teams and data scientists) to achieve self-service data integration, preparation, monitoring and delivery. At the heart of the platform are Data Connectors and Data Products (Nexsets), which can be autogenerated or manually created for user collaboration. Nexsets are logical entities that contain metadata related to datasets, data transformations, data access controls, data errors and alert configurations, to help accelerate and standardize data sharing and collaboration within an organization. Connecting Nexsets with different data systems generates ETL/ELT integration, streaming data integration, API integration, data APIs and more. At this time, Nexla does not meet our inclusion criteria for market presence, despite having a technically sound offering, and was therefore not included in the Magic Quadrant this year.
- **Striim** is headquartered in Palo Alto, California, and offers Striim Platform and Striim Cloud as its data integration tools. Striim Cloud is a unified stream data integration and stream analytics solution that operates as a fully managed service. Striim Platform offers a

self-hosted version of the same core software that can be deployed either in the cloud or on-premises. While other stream data integration tools mainly focus on data ingestion, with limited support for transformation, Striim is able to provide full support for stream data ingestion and integration, so it can be used for in-flight transformation, real-time enrichment and stream analytics. Customers praise Striim's log-based change data capture (CDC) capabilities, which are commonly used to stream data from on-premises and disparate cloud environments to cloud targets. In addition, Striim provides data delivery validation, metadata management, pipeline monitoring, event preview, and alerts, for continuous monitoring to ensure business SLAs and SLOs for real-time data delivery. Striim was excluded from this year's Magic Quadrant because it did not meet the inclusion criteria for revenue/customer count threshold or market presence.

Evaluation Criteria

Ability to Execute

Gartner analysts evaluate providers on the quality and efficacy of the processes, methods or procedures that enable IT provider performance to be competitive, efficient and effective, and to positively impact revenue, retention and reputation within Gartner's view of the market.

In this research, we evaluate the vendors' ability to execute in the data integration tool market by using the following criteria.

Product/Service

These are core goods and services that compete in and/or serve the defined market. This includes current product and service capabilities, quality, feature sets and skills. This can be offered natively or through OEM agreements/partnerships as defined in the market definition and detailed in the subcriteria.

We rated the vendors on:

- The vendors' capabilities that address current market requirements. These include but are not limited to bulk/batch data movement, CDC-based data replication and synchronization, data services orchestration, stream data integration for real-time use cases, data migration support, support for data engineering for analytics and data science, and other integration efforts for operational use cases (like MDM).

- The degree of openness of the vendor technology and product strategy – that is, the ability to exchange metadata with third-party offerings.
- The ability of offerings to allow interoperability to open-source solutions and third-party offerings. Some consumers are prepared to accept products from many different suppliers and assemble them together on their own. Interoperability is therefore appreciated by end users.
- Connectivity options to not only nonrelational databases, cloud applications and cloud data stores (such as cloud object stores and cloud data warehouses), but also traditional stores (including relational databases and enterprise applications).
- Connecting data integration activities to data quality and governance becomes integral in supporting those operational data integration use cases that require sharing high-quality data along with its lineage, such as master data management and B2B data sharing.
- The ability to offer both serverless metered pricing options (for net new use cases) and traditional pricing models such as node-/core-based models (when the use cases do not require flexibility of compute).
- The ability to support DataOps and orchestration capabilities to enable agile pipeline delivery, management, operationalization and maintenance. CI/CD integration, support for Git and Jenkins, support for regression test/validation, support for data observability, support for schema drift handling and so forth are all expected.

Overall Viability

Viability includes an assessment of the organization's overall financial health, as well as the financial and practical success of the business unit. This criterion views the likelihood of the organization to continue to offer and invest in the product as well as the product position in the current portfolio.

We rated the vendors on:

- The appropriateness of the vendor's financial resources, the continuity of its people and its technological consistency and how that affects the practical success of the business unit or organization in generating business results.
- The growth of the vendor's product lines, ARR, profitability and growth in new geographies/use cases.

- Product/services growth in revenue to determine the vendor growth in the data integration software market.
- Growth in high-momentum use cases such as cloud data integration. Revenue growth through cloud integration tools (iPaaS, SaaS, etc.).
- Other metrics to determine financial viability and spend on R&D efforts to continue differentiation and growth in product lines.
- Investment in skills, people and persona for product development, delivery and support. Retention and growth metrics – both are necessary.

Sales Execution/Pricing

This measures the vendor's capabilities in all presales activities and the structure that supports them. This includes deal management, pricing and negotiation, presales support, and the overall effectiveness of the sales channel.

We rated the vendors on:

- The ability of vendors to offer modular solutions. Organizations increasingly seek "modularity" or the capability to isolate specific required functions in data integration that are then reflected in their implementation approach and cost allocation.
- Ability to provide tools and capabilities through different pricing models appropriate by use cases, persona and environment is rated highly.
- The ability of vendors to support buyers that are looking for new pricing metrics that abstracts them from the underlying metrics of cloud pricing. Buyers are looking for vendor options that support serverless metered pricing metrics that are a true reflection of the actual work done and that can separate compute from storage/infrastructure.
 - Having said that, organizations are also wary that serverless metered pricing options that don't consider good financial governance can soon get out of control. Gartner will be closely evaluating vendors on their ability to enforce financial governance metrics into their pricing models and licensing metrics.
- On the ease with which customers can hold vendors accountable for agreed-upon SLAs. Buyers are evaluating ways through which they can hold vendors accountable for the promised SLAs (in terms of uptime, turnaround times to issues, bug fixes, migrations and

so on). Providers must demonstrate ways through which customers can escalate and attain credits/discounts when SLAs are not met.

- Finally, data management as a discipline needs to track, predict and preempt overall cost associated with cloud integration workloads. This is especially important as data and analytics teams are becoming distributed across various domains and are increasingly being placed in various lines of businesses. This makes it important for data management leaders to have the ability to associate the cost of running data integration workloads to the business value they provide. This can be done by analyzing performance and system metadata to best allocate processing capacity. Vendors will need to provide tools that can automate aspects of this through FinOps approaches.

Market Responsiveness/Track Record

This measures vendors' ability to respond, change direction, be flexible and achieve competitive success as opportunities develop, competitors act, customer needs evolve and market dynamics change. This criterion also considers the provider's history of responsiveness to changing market demands.

We rated the vendors on:

- We are looking for evidence on how the vendors "course-corrected" to changing buyer requirements in terms of use cases, upcoming capabilities, pricing models and support requirements.
 - As an example, managed services options for maintaining data pipelines and handling schema drifts are in demand, particularly by business teams and citizen integrators. Providers that can enable these requests are therefore selected over others that are still focused on IT teams alone.
- The requirements to enable data fabric designs are also increasing. We are looking for vendors that are adding features to enable more comprehensive data fabrics.
- Even though solutions that provide low-code/no-code UIs are preferred, we are also getting requests from data engineering teams for tools that support custom coding and importing scripts created in languages such as R and Python for highly advanced transformations.

- The market is also looking for vendors that can assist with moving away from environments that seem to have lost traction – for example, Hadoop – and toward modern sources and targets, such as cloud database platforms as a service and cloud applications.
- Vendors that support using AI/ML to automate complex and repetitive data integration tasks such as data transformation, orchestration, parts of data modeling and data delivery are preferred.
- Vendors that provide open ecosystems to support independent data integration (which does not depend on any cloud infrastructure specifically or a DBMS or a proprietary data exchange format or data storage format) are preferred.

Marketing Execution

This is the clarity, quality, creativity and efficacy of delivering the organization's message in order to influence the market, promote the brand, increase awareness of products and establish a positive identification in the minds of customers. This "mind share" can be driven by a combination of publicity, promotional activity, thought leadership, social media, referrals and sales activities.

We rated the vendors on:

- Brand recall value has a high premium in a mature market like data integration.
- Providers must develop a means of converting community "chatter" and excitement to support delivery and go-to-market campaigns.
- The overall effectiveness of the vendor's marketing efforts – which impact its mind share, market share and account penetration – is important.
- The ability of the vendor to adapt to changing demands in the market by aligning its product message with new trends and end-user interests was part of the evaluation.
- Suppliers need to be aware of emerging best practices for data management infrastructure and whether they and their customers can specifically benefit from specialized horizontal or vertical capabilities, geographically targeted approaches, or partner-supported implementation practices.
- Ability of the vendor to support and become a part of open community channels for best practices sharing, sharing connectors/code/mappings/other assets or support open metadata sharing standards.

- Finally, how the vendor is rated and perceived on community review and evaluation channels like the Gartner Peer Insights.

Customer Experience

These services should enable customers to achieve anticipated results. Specifically, this includes quality supplier/buyer interactions, technical support and account support. This may also include ancillary tools, customer support programs, availability of user groups and service-level agreements.

We rated the vendors on:

- We will evaluate the level of satisfaction expressed by customers with the vendor's product support and professional services support.
- We will also look at customers' overall relationship with the vendor, the experience while upgrading software versions, the learning curve for new users given the training resources available, and customer perceptions of the value of the vendor's data integration tools relative to cost and expectations.
- We will evaluate the ability of the vendor to maintain parity between cloud and on-premises software and measure after-sales support, migration ease, ease of deployment and overall maintenance and support.
- Customer feedback on the ability of their vendors to meet roadmap deliverables, technical knowledge sharing, skills enablement and augmentation, and training will all be evaluated.
- We will look at various platforms for such data points including but not limited to our interactions with end users in inquiries, Peer Insights data, surveys, customer reference calls, touchpoints across various Gartner and external events, community chatter, and vendor briefing data.
- The distinction between advanced use cases and "pedestrian" applications is becoming more pronounced. The evaluation this year is focused on separating success in "traditional" market delivery from success in "innovative" market delivery in reviewing the customer experience.

Operations

This is the ability of the organization to meet goals and commitments. Factors include quality of the organizational structure, skills, experiences, programs, systems and other vehicles that enable the organization to operate effectively and efficiently.

We rated the vendors on:

- Operations are not specifically differentiating to end-user markets, but product management consistency and support/maintenance practices add to the overall customer experience and to the stability of senior staff.
- Suppliers need to demonstrate a new balance in their R&D allocation to ensure they are positioned for deployment with greater focus on active metadata, metadata management overall and semantic tiers.
- Also, they must demonstrate that they can provide ongoing support for the massive bulk/batch data movement market and for other data delivery styles including replication, streaming, API, virtualization and messaging.
- Partner programs, skills augmentation, improvements in support and services, training materials and programs, and delivery with external service providers are all important in this evaluation criteria.

Table 1: Ability to Execute Evaluation Criteria

Evaluation Criteria ↓	Weighting ↓
Product or Service	High
Overall Viability	High
Sales Execution/Pricing	High
Market Responsiveness/Record	Medium

Evaluation Criteria ↓	Weighting ↓
Marketing Execution	Medium
Customer Experience	High
Operations	Low

Source: Gartner (August 2022)

Completeness of Vision

Market Understanding

This is the ability to understand customer needs and translate them into products and services. Vendors that show a clear vision listen, understand customer demands, and shape or enhance market changes with their added vision.

We rated the vendors on:

- The ability to formulate product vision around multicloud/intercloud and hybrid data integration capabilities. Also, how the vendors are participating in a cloud ecosystem
- The ability to provide core data integration services including all kinds of data movement topologies, support for bulk/batch, streaming, replication, API, messaging and support for the interoperating and combining of these data delivery styles if needed.
- The ability to provide “advisors” and other insights in design, development, deployment and management of integration services to support decision insights and decision automation.

- The ability to automate and augment data integration design and delivery through active metadata analysis and recommendation engines.
- The ability to provide business-user-friendly UIs through self-service data preparation modules and to allow skilled users to operationalize self-service findings and flows.
- The ability to work with data services through API management and integration and to deliver application and data integration flows together if the use case demands it. Existing support for delivery capabilities through microservices is a plus.
- The ability to contribute to data fabrics/mesh designs. We are looking for support for knowledge graphs, graph modeling, semantic modeling and enrichment, taxonomy to ontology mappings support, graph analysis, and graph query to support data fabrics. It would be interesting to view support for data product delivery through governance to distributed data product teams in domains.
- Ability to showcase capabilities (and upcoming ones on the roadmap) to support DataOps for agile and automated delivery of data integration pipelines. We will look for capabilities that support agile data engineering practices.
- Finally, the vision to support financial governance through plans for FinOps support and current capabilities to validate this.

Marketing Strategy

This is the clear, differentiated messaging consistently communicated through social media, advertising, customer programs and positioning statements.

We rated the vendors on:

- Ability to differentiate their offerings from various other categories in the market – for example, basic ingestion tools as well as integration tools supporting only open-source frameworks.
- Clear messaging on trial/freemium to full enterprise offerings (with differentiators across each support/SLA level)
- Good visibility into the product portfolio along with features across each separate tool – including possible overlaps, ways to buy, means to procure, support tiers and licensing – is now a must.

- Ability to showcase a complex portfolio through clear differentiated messaging justifying purchase and clarifying use of each product/SKU.
- Efforts and investments, with demonstrations, into partner programs, training programs, OEM/value-added reseller (VAR), other partnerships, cloud provider partnerships, SI partnerships and so on.
- Demonstrated proof of expansion in training, certifications and availability of talent in the market (through partner programs, training and so on).

Sales Strategy

This involves the strategy for selling and using the appropriate networks, including direct and indirect sales, marketing, service, and communication. This includes partners that extend the scope and depth of market reach, expertise, technologies, services and customer base.

We rated the vendors on:

- Expansion in sales partner networks
- Vendors' ability to become a part of different cloud ecosystems as an independent data integration partner for the customer
- Strategy to grow beyond existing markets, use cases, geographies, and core capabilities
- Demonstrated evidence of improvement in communication of existing and upcoming tools/services
- Affiliate partnerships
- Growth through varying channels (OEMs, VAR, Sis, consulting companies, joint go-to-market, partnerships with vendors in the data and analytics space, etc.)

Offering (Product) Strategy

This is an approach to product development and delivery that emphasizes market differentiation, functionality, methodology and features as they map to current and future requirements.

We rated the vendors on:

- Aligning existing tools and roadmaps with future market direction.
- We are looking for tools that can deliver distributed data integration across on-premises, cloud, intercloud and edge ecosystems.
- Tools must exhibit improvement in automation-oriented capabilities.
- While advanced capabilities are needed, the tools must not drop existing and “traditional” requirements of data integration, including bulk/batch capabilities, supporting hybrid/on-premises sources and targets, supporting developers and so on.
- There is now significant increased expectation on “active” metadata understanding, conversion, utilization and analysis:
- This active metadata is used in profiling, machine learning, evaluation of assets and comparison with existing integration upon connection.
- Self-correcting optimization in processes is now important and expected.
- Utilizing metadata to assist in user “push” recommendations for new data assets – and to create semantic knowledge graphs to assist with data fabric design that enables a more consistent (and application-neutral) semantic model for integration – is considered a differentiator.
- Given the requirement to support diverse environments for data, delivery models and platform-mix perspective, we assess vendors on the degree of openness of their technology and interoperability with other data and analytics tools.
- We will assess vendors’ roadmap and existing capabilities to support interoperability (through open metadata exchange) with other tools (their own or third party)
- We will also assess roadmaps to support DataOps and FinOps to support agile data engineering and cost optimization measures.
- Finally, we will be looking out for a roadmap (through demonstrable evidence) that supports seamless on-premises to cloud migration of tools or version/system migration in general. This will include changes to tool pricing as well.

Business Model

This is the design, logic and execution of the organization's business proposition to achieve continued success.

We rated the vendors on:

- The overall approach the vendor takes to execute on its strategy for the data integration tool market – including diversity of delivery models, packaging and pricing options, and partnerships – is important.
- It is both expected and reasonable to assume that tightly targeted models for traditional delivery needs can cut delivery cost, increase adoption and deliver specific integration needs to end-user organizations.
- The ability of the vendors to provide both “current” requirements through best-fit engineering tools versus future requirements through end-to-end platforms or best-of-breed options is a good measure of this category.
- The business proposition must include the ability for end-user organizations to try before they buy, and the tools must be able to interoperate with existing tools within the customer base rather than having to replace all current software.
- We will be looking out for vendors to create a niche for themselves in this complex market. How vendors carve out differentiation, land-expand, grow and target specific differentiated use cases, persona, delivery models and even operating models is evaluated.

Vertical/Industry Strategy

The strategy to direct resources (sales, product and development), skills and products to meet the specific needs of individual market segments, including verticals.

We rated the vendors on:

- We look at the degree of emphasis the vendor places on vertical solutions and the vendor's depth of vertical market expertise.
- Although most prospects are not looking for data integration tools focused on a specific vertical/domain because they rightly treat data integration as industry-agnostic, some organizations might favor tools that are able to create a specific industry model, ontology or knowledge graph based on an industry-specific taxonomy.
- Vertical/domain specific solution accelerators, KPIs, best practices and other industry starter templates might be favored by some buyers as well, in addition to industry/domain experts being a part of the professional services provided.

- The vendors' ability to provide vertical knowledge and expertise in presales, sales, implementation and support for targeted verticals is evaluated in this section.

Innovation

This is direct, related, complementary and synergistic layouts of resources, expertise or capital for investment, consolidation, defensive or preemptive purposes.

We rated the vendors on:

- The current innovation demands in the market are centered on managing location-agnostic capability in data integration – that is, the ability to not have to move or replicate data necessarily but to connect to data in-place when feasible and take the processing to the data (rather than vice versa) to execute integration.
- Integration should run on-premises and in the cloud, and switch between them.
- As data becomes highly distributed, data integration activities are also required to become easily distributable to any data location, and vendors should be able to recommend/determine when data needs to be moved for optimal processing and deliver workloads to the most optimal execution engines through containerized services and microservices architecture.
- Converging data and application integration approaches is now expected.
- ML-based automation using internal analytics on all kinds of collected metadata to support integration activities is another area of improvement that the market currently demands.
- The growing diversity of users indicates a much higher demand for administrative, auditing, monitoring and even governance controls that utilize job audit statistics.
- Graph analysis to determine user classification and optimization “hints” are also increasingly demanded.
- Provide support for financial governance to enable business units, CDOs and CFOs to support cost optimization as workloads move to the cloud.

- Finally, because the increase in the number of data pipelines is inevitable, organizations are expecting DataOps-oriented capabilities that can support CI/CD; project management capabilities such as Git and Jenkins; automated testing/validation; the handling of various environments in an agile manner; sandboxes on demand and management of them; and agile pipelines creation, reuse, execution, management and so on.

Geographic Strategy

The provider's strategy is to direct resources, skills and offerings to meet the specific needs of geographies outside the "home" or native geography – either directly or through partners, channels and subsidiaries – as appropriate for that geography and market.

We rated the vendors on:

- The ability for vendors to provide their customers with local support with differing levels of confidence in the various approaches possible (that is, VARs, resellers, channel partners, OEM offerings and distributors).
- The ability to provide continuity of support across regions.
- The ability for development platforms to monitor where data originates with jurisdictional cognizance, and where it is eventually delivered.
- Their ability to address the possible violation of national laws due to data movement.
- The vendor's strategy for expanding into markets beyond its home region/country and its approach to achieving global presence (for example, direct local presence and use of resellers/distributors) are crucial for capitalizing on global demands for data integration capabilities and expertise.
- Level and performance of support in different geographies and skills availability to support after sales maintenance, etc.

Table 2: Completeness of Vision Evaluation Criteria

Evaluation Criteria ↓	Weighting ↓

<i>Evaluation Criteria</i> ↓	<i>Weighting</i> ↓
Market Understanding	High
Marketing Strategy	High
Sales Strategy	Medium
Offering (Product) Strategy	High
Business Model	Medium
Vertical/Industry Strategy	Low
Innovation	High
Geographic Strategy	Medium

Source: Gartner (August 2022)

Quadrant Descriptions

Leaders

Leaders are front-runners in their capability to support the combination of different data delivery styles (for example, the ability to combine and switch between ETL/ELT, replication, messaging, API integration and data virtualization based on their use-case demands).

Leaders exhibit significant market mind share and recognize the need for new and emerging market demands – often providing new functional capabilities in their products ahead of demand – by identifying new types of business problems to which data integration tools can bring significant value. Leaders have an established market presence, significant size and a multinational presence.

In 2022, Leaders in this market have started delivering on the data fabric promise – that is, their ability to balance collecting data with connecting to data. They automate the process of collecting all types of metadata (not just passive) and then represent the metadata in a graph to preserve context. This is then followed by improving the data modeling process by enriching the models with agreed-upon semantics. Finally, some vendors embed AI/ML toolkits, which utilize active metadata (as input) to start automating various aspects of data integration design and delivery. Most vendors in the Leaders quadrant provide capabilities to deliver the data fabric, although some require customization.

Leaders are adept at providing tools that can support both hybrid integration and multicloud integration options, bridging the data silos that exist across on-premises and multicloud ecosystems. Leaders allow organizations to remain independent in data integration as they look to deploy workloads across multiple CSPs, and they allow organizations to effectively provision cloud ecosystems.

Leaders provide efficient data engineering through self-service data preparation capabilities and integration portability. They also provide the ability to deliver pipelines and code through containerized services. Leaders identify the need for financial governance, especially for integration workloads running in the cloud and support needs to balance flexibility with cost optimization.

Leaders are strong in establishing their data integration tools to support both traditional and new data integration patterns to capitalize on market demand.

Challengers

Bulk/batch delivery styles (such as replication, streaming or data virtualization) are no longer differentiating but a “must have” feature. As such, this capability is now considered under execution rather than vision. In line with this market shift, Challengers have been making significant strides in delivering these capabilities within a broader metadata-driven data integration toolset.

In 2022, the Challengers listed in this research exhibit a strong understanding of the current data integration market demand and exhibit both the credibility and viability to deliver. Some Challengers are mature in specific core capabilities, which enables them to deliver targeted use cases faster and with a better overall TCO than other vendors. These vendors have developed best practices for leveraging their strongest product capability in new delivery models. For example, they can productize and market (1) data replication as a key strength for targeted use cases such as cloud data migration, data warehouse automation and data lake enablement or (2) data virtualization for faster turnaround time to analytics or (3) extensive support for moving data and workloads from nonrelational DBMSs such as Hadoop to cloud data warehouses or cloud data lakes.

Challengers generally have substantial customer bases. They exhibit strong market presence, although implementations may be of a single-project nature or reflect multiple projects of a single type – for example, predominantly data replication or application integration, or use cases specific to mainframe data, analytics/BI or data science. Gartner realizes that many customers have specialized demands for their most urgent or upcoming projects. We also recognize vendors in the Challengers quadrant that (if needed) can scale to support broader data integration use cases but can also customize their offerings for specific/traditional use cases, data types, data sources/targets, execution environments or specific CSPs.

Overall, the market is pushing Challengers to embrace the market vision of a data fabric, automated data integration and multicloud/hybrid data integration to deliver solutions that can automate various data integration tasks. These tasks include automated profiling, automation of repetitive transformations, data preparation (and the ability to use ML to automate self-service data preparation), performance optimization, query optimization, and movement of workloads to data stores and engines that are best suited for processing. Overall, this move toward enabling data fabric architectures with support for financial governance automation through FinOps is a key driver that will determine which Challengers can move into the Leaders quadrant next year.

Visionaries

Visionaries demonstrate a strong understanding of emerging technology and business trends or focus on a specific market need that is far outside of common practices, while also possessing capabilities that are expected to grow in demand. In 2022, the Visionaries in this Magic Quadrant have focused early on futuristic capabilities and go-to-market strategies to capitalize on their capacity to leverage:

- Augmented data integration through the data fabric design
- Representing data and metadata in business taxonomies and ontologies with support for semantic modeling
- The ability to provision a knowledge graph of data entities that represent multirelationship data in business context

- Serverless integration that supports a multicloud and hybrid cloud integration architecture
- The delivery of various data integration functionalities as loosely coupled API/microservices
- Seamless orchestration through DataOps techniques
- iPaaS delivery models led by the convergence of data integration, application integration and API integration/API management use cases
- Deliver data applications as data products and support federated governance.

In addition, a significant driver of vision is the ability of tools to connect to and analyze all forms of metadata — both passive and, increasingly, active metadata. With this, tools provide key statistics to developers and citizen integrators that aid with integration design and automation. Visionaries should lead the push toward the utilization of graphs, semantics, knowledge graphs and AI/ML for significant automation in data integration design, delivery and maintenance. Visionaries sometimes lack market mind share or credibility beyond their established customer base, their main use cases or very specific application domains/verticals. They may lack partnerships and (or) tight integrations with other incumbent data management vendors including third-party metadata management, data governance and data quality solutions. Visionaries could still be ramping up partnerships with system integrators, consulting companies and other partners that can ensure consistent support, implementation, training or after-sales support to their clients worldwide and beyond their main established markets. They may lack the installed base and global presence of larger vendors. Finally, Visionaries may be established players in related markets that have only recently placed an emphasis on data integration.

Niche Players

Because this is a mature market, Niche Players generally do not generally exhibit gaps in primary functionality or features. Instead, they are simply challenged to improve their execution or have not identified a specific market approach that expands use cases for their technology. This means that almost every Niche Player will be able to deliver against standard market expectations, both in functionality and cost/price options.

Niche Players may not appear very frequently in competitive situations for comprehensive and/or enterprise-class data integration deployments. Many have very strong offerings for a specific range of data integration problems — for example, a set of specific data delivery styles (batch, replication, streaming or virtual), application domains or use-case scenarios — and deliver substantial value for their customers in the associated segment. Niche Players often exhibit advantages in pricing in their small footprint and in vertical or

horizontal solutions and can't complement an organization's other data management technologies. Niche Players are known for solving one part of the data integration problem well through a targeted solution and may be a "good enough" solution for organizations with less complex needs.

Importantly, Niche Players in this market have demonstrated their capability to outperform dozens of tool and solution offerings that were considered and eventually excluded from this Magic Quadrant, but may be lacking maturity on certain features that display market vision.

Context

The market for data integration tools continues to evolve and is supported by strong levels of consolidation, strong revenue growth of 11.8% in 2021 (to reach \$3.8 billion at the end of 2021) and rapid adoption. More data and analytics leaders are realizing that data integration is a critical component of their data management infrastructure. They understand that they need to employ data integration functions to share data across all organizational and systemic boundaries. Therefore, organizations are increasingly seeking a comprehensive range of improved data integration capabilities to modernize their data, analytics and application infrastructures.

Data and analytics leaders must navigate a market brimming with products that claim to solve a range of data integration problem types. However, not all vendor solutions have experience in, or even provide, all the relevant capabilities needed across our key data integration use cases (see the companion [Critical Capabilities for Data Integration Tools](#)). Some vendors focus heavily on providing solutions focused on just one data integration style such as bulk/batch (through ETL or ELT), data replication (through CDC), messaging (through Kafka), or virtual (through data virtualization). But they may place less emphasis on the important capability of interoperating, orchestrating or even combining these different data delivery styles (ETL with data virtualization or data replication or messaging through API integration, for example) for accomplishing key use cases.

Some organizations have determined that basic functions are adequate and are seeking tools with targeted capabilities. As a result, they are interested in tools that are specialists in one data delivery style (for example, data replication, data ingestion, API integration, self-service data preparation or data virtualization). Some organizations prefer tools that can support one use case (such as cloud data ingestion and migration), one data type (such as IoT data integration) or one scenario (such as location intelligence through geospatial data integration focus). Such organizations can confidently start with the vendors in the Niche Players.

Organizations that seek tools that are generalists and can support multiple use cases through a combination of different data integration styles should evaluate the vendors mentioned in the Challengers and Leaders quadrants.

In addition, vendors in the Leaders quadrant are focused on automating various aspects of data integration. These include design, ingestion, schema mapping, schema drift detection and corrections, dataOps automation, orchestration automation, next-best transforms, automated lineage and impact analysis, and infrastructure management. These capabilities for augmented data integration demand a new data integration design — one that supports a balance of connect and collect data integration strategies.

Active-metadata-enabled data integration augmentation is a significant driver of market vision this year. Metadata as a byproduct of the design and operations management of a data integration platform is a minimum requirement of data integration tools in 2022. Platforms and solutions are now expected to provide continuous feedback regarding the profiles, quality, location, performance optimization, lineage, use cases, access points, context, frequency of access and content analysis of integrated data assets. As far as architects and solution designers are concerned, this feedback was long overdue. It is expected that graph analytics, powered by every conceivable type of metadata, will provide the necessary data fabric designs for introducing ML capabilities into data integration platforms (see [How to Activate Metadata to Enable a Composable Data Fabric](#)). This capability for active-metadata-based integration has been weighted very highly to define the vision of the market this year by Gartner analysts.

Gartner sees that the need to acquire and integrate data across multiple CSPs, typically for hybrid cloud and intercloud integration, is becoming crucial to many data integration use cases. Vendors that support just one cloud provider (CSP) or only their own databases or applications will fall behind due to valid lock-in and CSP dependence concerns.

An interesting trend from our inquiries revealed that an increasingly high number of data and analytics leaders are investigating and adopting tools that can support data ingestion and replication. This increase is because organizations are looking to ingest or replicate the data from their operational DBMSs to cloud data warehouses. This has been a significant driver of growth for many data integration providers (such as Fivetran, Matillion, Qlik and Software AG [StreamSets]). These providers have formed significant partnerships with CSPs like AWS, GCP and Microsoft Azure, along with popular cloud data warehouse vendors such as Snowflake and Teradata, to deliver integrated data to cloud data warehouses and data lakes for analytics. This task is often termed as data warehouse automation and is seen as a differentiator by organizations evaluating tools.

This year, organizations have also started evaluating vendors for their ability to support integration portability and improve the ability to deliver integration flows, mappings, assets and pipelines in an agile and orchestrated manner. Support for DataOps has been rated as a significant driver for vision by our analyst team. This is in line with market demand to support infrastructure as code and to enable the ability to port integration pipelines for optimal executions to environments that best support the required SLAs. Organizations now expect vendors to deliver an increasing array of such capabilities embedded in solutions ranging from support for CI/CD and integration with Git and Jenkins to providing automated serverless execution capabilities. Support for open-source data transformation

tools (like DBT), orchestration solutions (like Apache Airflow), and open-source ingestion capabilities using crowdsourced connectors (from vendors like Airbyte, for example) are a sign of vision for this edition of the Magic Quadrant.

For the first time, we have also started evaluating vendors on their ability to iteratively control spending, understand product performance, and help make choices regarding price-to-performance trade-offs, resulting in optimal allocation of resources in the cloud. This is what we are calling FinOps, and it's vital to address the growing concerns of overspending, overbudgeting and cost optimization in the cloud.

As more and more subject matter experts become a part of data integration tasks, their needs should be better supported through native integration with self-service data preparation tools and modules. These modules should support less technically skilled personnel with low-code/no-code integration environments. This capability is now a must-have. Moreover, vendors must also support data engineers who are tasked with operationalizing self-service models to production environments after guaranteeing governance and compliance.

Finally, a mix of data integration approaches has remained crucial, spanning from physical delivery to virtualized delivery, and from bulk/batch movements to event-driven granular data propagation. When data is being constantly produced in massive quantities and is always in motion and constantly changing (for example, IoT platforms and data lakes), attempts to collect all this data are neither practical nor viable. This is driving an increase in demand for connection to data, not just the collection of it (see [Assessing the Relevance of Data Virtualization in Modern Data Architectures](#)). In 2022, data virtualization has again been a key criterion for evaluating vendors. In addition, we see a surge in demand for tools that can integrate "data in motion" through stream data integration technologies that provide native connectors to event data sources (like clickstreams, logs and IoT) and the ability to capture, enrich and deliver event data to data stores/applications for real-time analytics and other use cases (see [Market Guide for Event Stream Processing](#)).

Market Overview

The data integration tools market continues to push toward distributed and augmented data management. This push is inherent in the modern data fabric architecture (see [Data and Analytics Essentials: Data Fabric](#)). *The market has realized that those data integration tools that do not balance "collect"- with "connect"-based data management architectures will always result in data silos and/or poorly integrated infrastructures.* Moving forward, organizations will need to monitor trends that are affecting enterprise requirements and vendor offerings. We highlight some of these below:

- **Data integration market experiences double-digit growth:** The market grew at 11.8% in 2021 as compared with 6.8% in 2020 (see [Market Share: Data Integration Software, Worldwide, 2021](#)). The increased growth in this market is part of the broader trend of postpandemic recovery reflected in higher growth of software markets overall, as software growth increased from 9.0% in 2020 to 16.0% in 2021. This increased growth in 2021 for the data integration market is reflected in our updated forecasts for growth as well. Five-year compound annual growth rate (CAGR) for the 2021 to 2026 time frame forecast is now 8.5% (see [Forecast: Enterprise Infrastructure Software, Worldwide, 2020-2026, 2Q22 Update](#)), updated from 7.0% same time last year. Cloud adoption continues to be significant, with the iPaaS market growing by 40.2% in 2021.
- **Market leaders continue to lose ground to smaller vendors:** The top five vendors in this market (based on their market share) had a collective market share of 71% in 2017. This number has been steadily dropping over the years, and in 2021, the collective market share was only 52%. A similar trend can be seen when analyzing the top three or even top 10 vendors. One of the main reasons for this is that market share leaders such as Informatica and Talend are ceding market share as they focus their growth efforts primarily on their iPaaS products. Another reason is that smaller vendors with more focused offerings (such as those targeted toward a specific data integration capability like data ingestion or data virtualization etc) continue to disrupt larger vendors with their land-and-expand strategies.
- **Market growth is being driven by support for modern data delivery styles and cloud data ecosystems:** Vendors gaining market share have a common theme: They focus on leadership in specific data integration styles such as data virtualization or data replication, and/or they focus on data integration delivered as a native and managed cloud service. Some high-growth vendors in the first category are Denodo, Qlik, Software AG (StreamSets), Striim and Confluent. These vendors collectively grew their revenue by 32% in 2021. Some high-growth vendors in the second category are AWS, Fivetran, Google, Matillion and Microsoft. These vendors collectively grew their revenue by 94% in 2021. Although these growth numbers are off a much smaller revenue base, the gradual decline in market share for the larger and established vendors shows that they will need to find the right balance between all-encompassing platform solutions and easily accessible point solutions to keep pace (see [Market Share Analysis: Data Integration Software, Worldwide, 2021](#)). Some providers have focused on vendor acquisitions to gain maturity in specific data delivery styles in the last couple of years. The most prominent examples include Fivetran acquiring HVR Software (a log-based data replication vendor), Qlik acquiring Blendr.io (for application integration), Talend acquiring Gamma Soft (a log-based change data capture vendor) and TIBCO Software acquiring Information Builders (a batch ETL/ELT technology provider).
- **Data fabric is critical and driven by the end-user push toward augmented data integration:** Another huge completeness of vision criterion that the market is demanding is augmented data integration design and delivery. The COVID-19 pandemic has only fast-

tracked this strategic direction of the market. Data and analytics leaders are realizing that they cannot keep investing in manual data integration; they need automation support. Data integration teams (in terms of individual members) are constantly contracting — the median number of individuals in teams is less than 10 (based on anecdotal evidence from our inquiries). And while team sizes are reducing, the amount of data and, hence, the number of data integration requirements, are growing exponentially. This gap between demand and supply is pointing toward an urgent focus on automation and augmentation. Augmented data integration demands a renewed focus on the data fabric architecture design, which is a key use case for this year. The data fabric is an architecture pattern that informs and automates the design, integration and deployment of data objects. This approach results in faster, informed and, in some cases, completely automated data access and sharing. In 2021, some vendors have been able to combine most (if not all) of these capabilities needed to deliver the data fabric into productized solutions, which signifies leadership. Others are going in this direction through partnerships, merger and acquisition (M&A) activity, product enhancements and, more frequently, a combination of all these (see [Quick Answer: What Is Data Fabric Design?](#)).

- **Data mesh starts gaining traction:** One of the key reasons why organizations investigate the data mesh design is that they require decentralized and domain-oriented data delivery of “data products.” This push is gathering pace because business teams believe that they are often left waiting for data engineering resource attention and that the data pipelines delivered by engineering teams do not do justice to their time-to-market SLAs. Business teams also request more data integration and modeling ownership to impart subject matter expertise into their data products through semantic modeling, ontology creation and knowledge graph support. As a result, organizations are actively evaluating data integration tools that can balance centralized data pipeline delivery with technical capabilities to provision decentralized data product delivery through capabilities such as data-as-a-service, DataOps-based CI/CD, infrastructure-as-code, GIT integration, scheduling and testing automation, and other agile capabilities to deliver data as a product to domains and business teams. See [Data Fabric or Data Mesh: How to Decide Your Future Data Management Architecture](#).
- **FinOps and financial governance are to be taken seriously for cost optimization in cloud data integration efforts:** This is the first time that Gartner is introducing FinOps as a key capability for evaluating vendors in this Magic Quadrant research. Data integration tools need to track, predict and preempt overall cost associated with cloud integration workloads as data and analytics teams get distributed across various domains and are being placed increasingly in various lines of businesses. This makes it important for data and analytics leaders to have the ability to associate cost of running data integration workloads to the value associated with them and have control over allocating processing capacity to workloads they deem to be important through optimal analysis of performance and system metadata and the ability to associate value to cost. Vendors will therefore need to provide tools that can enable financial governance and automate aspects of this through FinOps approaches that also reflect in their pricing models and governance models. FinOps for data integration comprises the capabilities that enable data and analytics leaders to iteratively

control spending, understand product performance and make choices regarding price-to-performance trade-offs, resulting in optimal allocation of resources in the cloud for efficient cost optimization.

- **Data engineering requirements are fueling the next round of investments:** Data engineering is the discipline of translating data into usable forms by building and operationalizing data pipelines across various data and analytics platforms. It goes beyond the traditional data management practices to include software engineering and infrastructure operations practices (see [How to Build a Data Engineering Practice That Delivers Great Consumer Experiences](#)). An example is using coding languages like Python and Scala to automate data pipeline builds, regression tests, deployments and operations monitoring. With more and more data infrastructure running on the cloud, platform operations are becoming a core part of data teams' responsibilities. The market around data engineering is still emerging, and there are no set industrywide standards, which limits the use-case application. Data integration tools are stepping up to this need and providing various built-in capabilities to assist end-user customers. Therefore, organizations prefer those data integration tools that embed capabilities that assist data engineers to build, manage, operationalize and even retire data pipelines in an agile and trusted manner, as well as run their pipelines in various execution environments. Data integration tools that allow optimizing code and pipeline execution through pushdowns, containerizations and serverless execution are being preferred in competitive RFPs.
- **DataOps needs to be supported:** Even though data integration tools don't by themselves provide all capabilities necessary to deliver DataOps, they certainly support DataOps enablement. DataOps is not a single tool or process but is instead a focus on building collaborative workflows to make data delivery more predictable. For additional information, see [Data and Analytics Essentials: DataOps](#). Those data integration tools that can support DataOps are naturally being favored over those that do not. Based on our inquiries with clients, key aspects being requested include the ability to deliver data pipelines through CI/CD. Organizations also request capabilities that support automated testing and validation of code. Leading tools also provide the ability to integrate their tools with project management and version-control tooling like GIT, Jenkins and Maven. Data engineering departments in end-user organizations prefer data integration tools that can help them balance the low-code capabilities (offered as part of the data integration tool) with options to invoke code developed in external Python libraries; in dbt (for complex transformation where manual coding is preferred); and in Apache Airflow, Luigi, Prefect, Dagster and so on for task workflow scheduling, for example. Some data integration tools also enable organizations to manage different nonproduction environments (such as sandbox, development and test/quality assurance [QA]) in an agile manner.
- **Support for hybrid and intercloud data management is now critical:** Cloud architectures for data management span hybrid, multicloud and intercloud deployments. There are both risks and benefits in managing data across diverse and distributed

deployment environments. Data location impacts performance, data sovereignty, application latency SLAs, high-availability and disaster recovery strategies, and financial models. Gartner estimates nearly half of data management implementations use both on-premises and cloud environments. Hybrid data management and integration support has therefore become pivotal in the market. Data integration tools are expected to dynamically construct or reconstruct integration infrastructure across a hybrid data management environment. Those tools that can support integrating data across different cloud infrastructures and synchronize it with on-premises data sources and targets have been given more vision scores in this year's Magic Quadrant revision.

- **Independent data integration tools are needed to prevent application/cloud service provider (CSP)/database lock-in:** There needs to be a clear focus on independent data integration tools that do not necessitate the movement and persistence of data into a specific vendor repository or cloud ecosystem. This is more important than ever because embedded data integration capabilities delivered by vendors as part of a broader application (such as analytics and BI or CRM tool) or database, or even CSP-specific data integration solutions, might make it easy for organizations to ingest data into one database, application or CSP ecosystem. However, these same embedded integration capabilities do very little to allow organizations to integrate data across different data stores, applications or multicloud/hybrid environments. This could lead to potential vendor lock-in challenges, high egress costs and data silos, resulting in the inability of organizations to reuse integrated data for general-purpose use cases. Although the CSP native data integration tools have started becoming more open in terms of allowing for two-way integration (to and from their own cloud data stores), those organizations that are looking to invest in a general-purpose (and independent) data integration tool for use cases involving more than one cloud service provider must favor independent data integration tools to partner with those provided by CSPs, DbPaaS or even their analytics/BI or data science vendors.

Evidence

The analysis in this Magic Quadrant research is based on information from several sources, including:

- An RFI process that engaged vendors in this market. It elicited extensive data on functional capabilities, customer base demographics, financial status, pricing and other quantitative attributes.
- Interactive briefings in which vendors provided Gartner with updates on their strategy, market positioning, recent key developments and product roadmap.
- Feedback about tools and vendors captured during conversations with users of Gartner's client inquiry service.

- Market share and revenue growth estimates developed by Gartner's technology and service provider team. Peer feedback from Gartner Peer Insights, comprising peer-driven ratings and reviews for enterprise IT solutions and services covering more than 300 technology markets and 3,000 vendors.

Evaluation Criteria Definitions

Ability to Execute

Product/Service: Core goods and services offered by the vendor for the defined market. This includes current product/service capabilities, quality, feature sets, skills and so on, whether offered natively or through OEM agreements/partnerships as defined in the market definition and detailed in the subcriteria.

Overall Viability: Viability includes an assessment of the overall organization's financial health, the financial and practical success of the business unit, and the likelihood that the individual business unit will continue investing in the product, will continue offering the product and will advance the state of the art within the organization's portfolio of products.

Sales Execution/Pricing: The vendor's capabilities in all presales activities and the structure that supports them. This includes deal management, pricing and negotiation, presales support, and the overall effectiveness of the sales channel.

Market Responsiveness/Record: Ability to respond, change direction, be flexible and achieve competitive success as opportunities develop, competitors act, customer needs evolve and market dynamics change. This criterion also considers the vendor's history of responsiveness.

Marketing Execution: The clarity, quality, creativity and efficacy of programs designed to deliver the organization's message to influence the market, promote the brand and business, increase awareness of the products, and establish a positive identification with the product/brand and organization in the minds of buyers. This "mind share" can be driven by a combination of publicity, promotional initiatives, thought leadership, word of mouth and sales activities.

Customer Experience: Relationships, products and services/programs that enable clients to be successful with the products evaluated. Specifically, this includes the ways customers receive technical support or account support. This can also include ancillary tools, customer support programs (and the quality thereof), availability of user groups, service-level agreements and so on.

Operations: The ability of the organization to meet its goals and commitments. Factors include the quality of the organizational structure, including skills, experiences, programs, systems and other vehicles that enable the organization to operate effectively and efficiently on an ongoing basis.

Completeness of Vision

Market Understanding: Ability of the vendor to understand buyers' wants and needs and to translate those into products and services. Vendors that show the highest degree of vision listen to and understand buyers' wants and needs, and can shape or enhance those with their added vision.

Marketing Strategy: A clear, differentiated set of messages consistently communicated throughout the organization and externalized through the website, advertising, customer programs and positioning statements.

Sales Strategy: The strategy for selling products that uses the appropriate network of direct and indirect sales, marketing, service, and communication affiliates that extend the scope and depth of market reach, skills, expertise, technologies, services and the customer base.

Offering (Product) Strategy: The vendor's approach to product development and delivery that emphasizes differentiation, functionality, methodology and feature sets as they map to current and future requirements.

Business Model: The soundness and logic of the vendor's underlying business proposition.

Vertical/Industry Strategy: The vendor's strategy to direct resources, skills and offerings to meet the specific needs of individual market segments, including vertical markets.

Innovation: Direct, related, complementary and synergistic layouts of resources, expertise or capital for investment, consolidation, defensive or pre-emptive purposes.

Geographic Strategy: The vendor's strategy to direct resources, skills and offerings to meet the specific needs of geographies outside the "home" or native geography, either directly or through partners, channels and subsidiaries as appropriate for that geography and market.

**Learn how Gartner
can help you succeed**

Become a Client

© 2022 Gartner, Inc. and/or its affiliates. All rights reserved. Gartner is a registered trademark of Gartner, Inc. and its affiliates. This publication may not be reproduced or distributed in any form without Gartner's prior written permission. It consists of the opinions of Gartner's research organization, which should not be construed as statements of fact. While the information contained in this publication has been obtained from sources believed to be reliable, Gartner disclaims all warranties as to the accuracy, completeness or adequacy of such information. Although Gartner research may address legal and financial issues, Gartner does not provide legal or investment advice and its research should not be construed or used as such. Your access and use of this publication are governed by [Gartner's Usage Policy](#). Gartner prides itself on its reputation for independence and objectivity. Its research is produced independently by its research organization without input or influence from any third party. For further information, see "[Guiding Principles on Independence and Objectivity](#)."

About Careers Newsroom Policies Site Index IT Glossary Gartner Blog Network Contact Send Feedback 

© 2022 Gartner, Inc. and/or its Affiliates. All Rights Reserved.



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Data Partitioning

Pravin Y Pawar

Adapted from Microsoft "[Data partitioning guidance](#)"

Data partitioning guidance

- In many large-scale solutions, data is divided into **partitions**
 - can be **managed and accessed separately**
- Partitioning can
 - **improve scalability,**
 - **reduce contention,**
 - **and optimize performance**
- Can also provide a mechanism for **dividing data by usage pattern**
 - can archive older data in cheaper data storage
- However, the partitioning strategy must be chosen carefully to maximize the benefits while minimizing adverse effects

Why partition data?

Reasons

- Improve scalability
 - When a single database system scaled up, it will eventually reach a physical hardware limit
 - If data is divided across multiple partitions, each hosted on a separate server, can scale out the system almost indefinitely
- Improve performance
 - Data access operations on each partition take place over a smaller volume of data
 - Correctly done, partitioning can make system more efficient
 - Operations that affect more than one partition can run in parallel
- Improve availability
 - Separating data across multiple servers avoids a single point of failure
 - If one instance fails, only the data in that partition is unavailable, operations on other partitions can continue
- Improve security
 - In some cases, sensitive and non-sensitive data can be separated into different partitions and apply different security controls to the sensitive data

Designing partitions

Three typical strategies for partitioning data

- Horizontal partitioning (aka sharding)
 - each partition is a separate data store, but all partitions have the same schema
 - each partition is known as a shard and holds a specific subset of the data
 - such as all the orders for a specific set of customers
- Vertical partitioning
 - each partition holds a subset of the fields for items in the data store
 - fields are divided according to their pattern of use
 - frequently accessed fields might be placed in one vertical partition and less frequently accessed fields in another
- Functional partitioning
 - data is aggregated according to how it is used by each bounded context in the system
 - an e-commerce system might store invoice data in one partition and product inventory data in another
- These strategies can be combined
 - might divide data into shards and then use vertical partitioning to further subdivide the data in each shard

Horizontal partitioning (sharding)

Example Product data

- Product inventory data is divided into shards based on the product key
- Each shard
 - holds the data for a contiguous range of shard keys (A-G and H-Z) organized alphabetically
- Sharding spreads the load over more computers, which reduces contention and improves performance

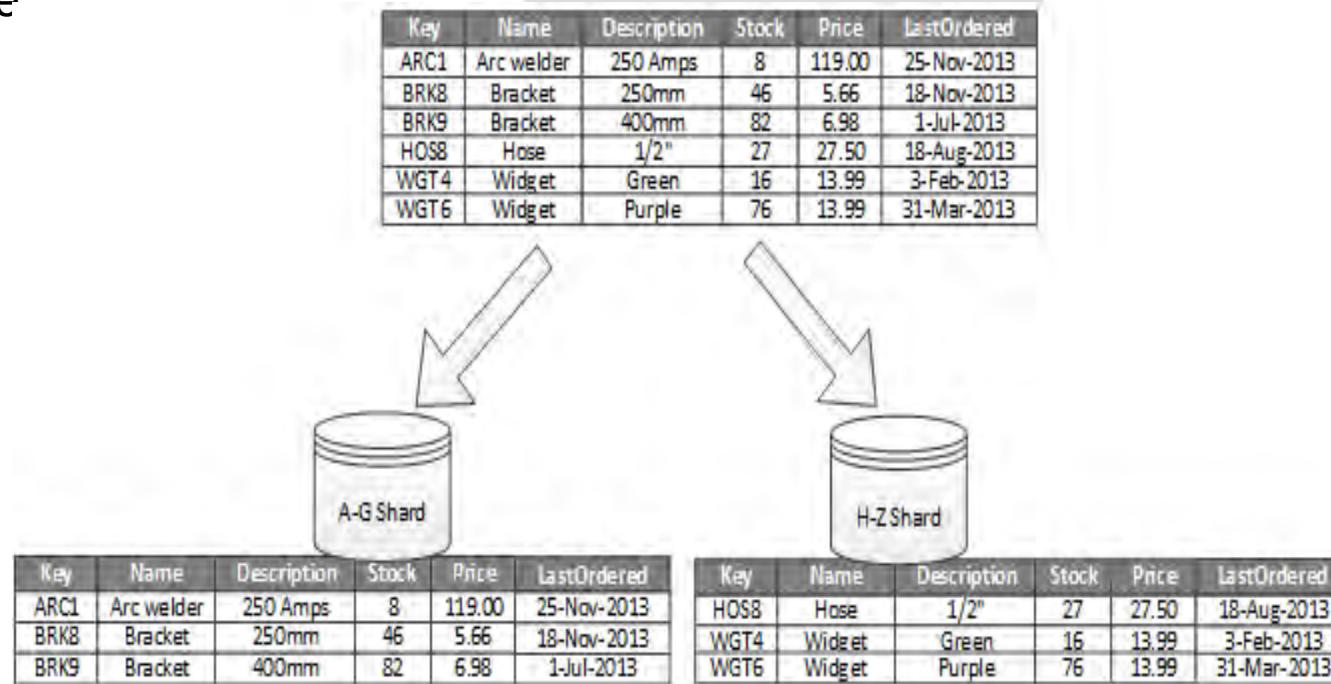


Figure - Horizontally partitioning (sharding) data based on a partition key.

Horizontal partitioning (sharding) - 2

Practices

- Sharding key
 - most important factor is the **choice of a sharding key**
 - can be **difficult to change the key** after the system is in operation
 - key must ensure that **data is partitioned to spread the workload as evenly as possible across the shards**
 - Choose a sharding key that minimizes any future requirements to split large shards, coalesce small shards into larger partitions, or change the schema can be **very time consuming**, and might require taking one or more **shards offline** while they are performed
- Size
 - The **shards don't have to be the same size**
 - more important to **balance the number of requests**
 - Some shards might be very large, but each item has a low number of access operations
 - Other shards might be smaller, but each item is accessed much more frequently
- Access
 - **Avoid creating "hot" partitions** that can **affect performance and availability**
 - using the first letter of a customer's name causes an unbalanced distribution, because some letters are more common
 - instead, use a hash of a customer identifier to distribute data more evenly across partitions
- Replication
 - If shards are replicated, it might be **possible to keep some of the replicas online while others are split, merged, or reconfigured**
 - the system might need to limit the operations that can be performed during the reconfiguration
 - the data in the replicas might be marked as read-only to prevent data inconsistencies

Vertical partitioning

Example of vertical partitioning

- Different properties of an item are stored in different partitions
 - One partition holds data that is accessed more frequently, including product name, description, and price
 - Another partition holds inventory data: the stock count and last-ordered date

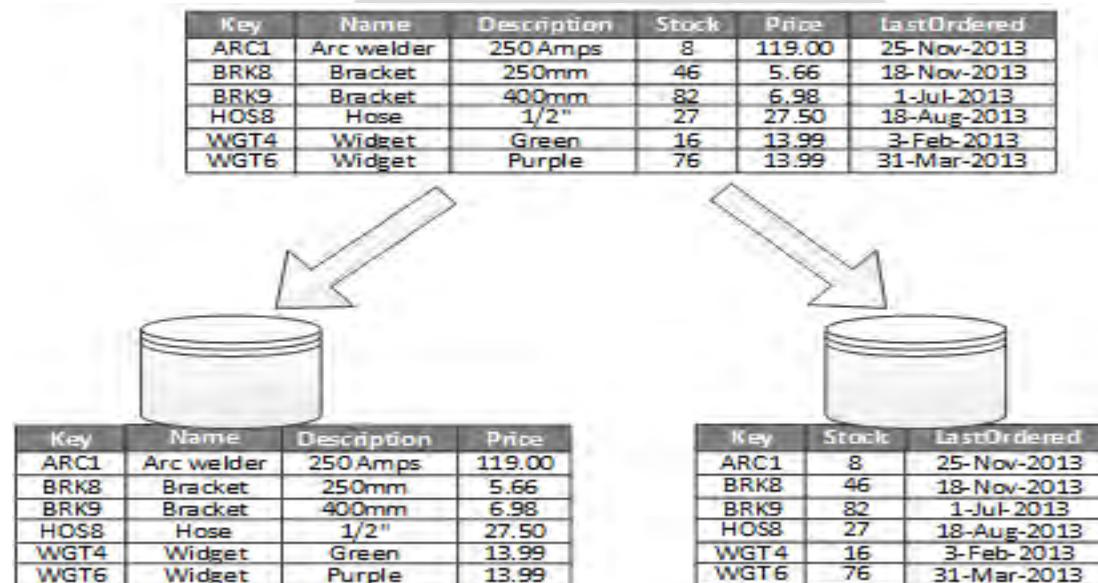


Figure . - Vertically partitioning data by its pattern of use.

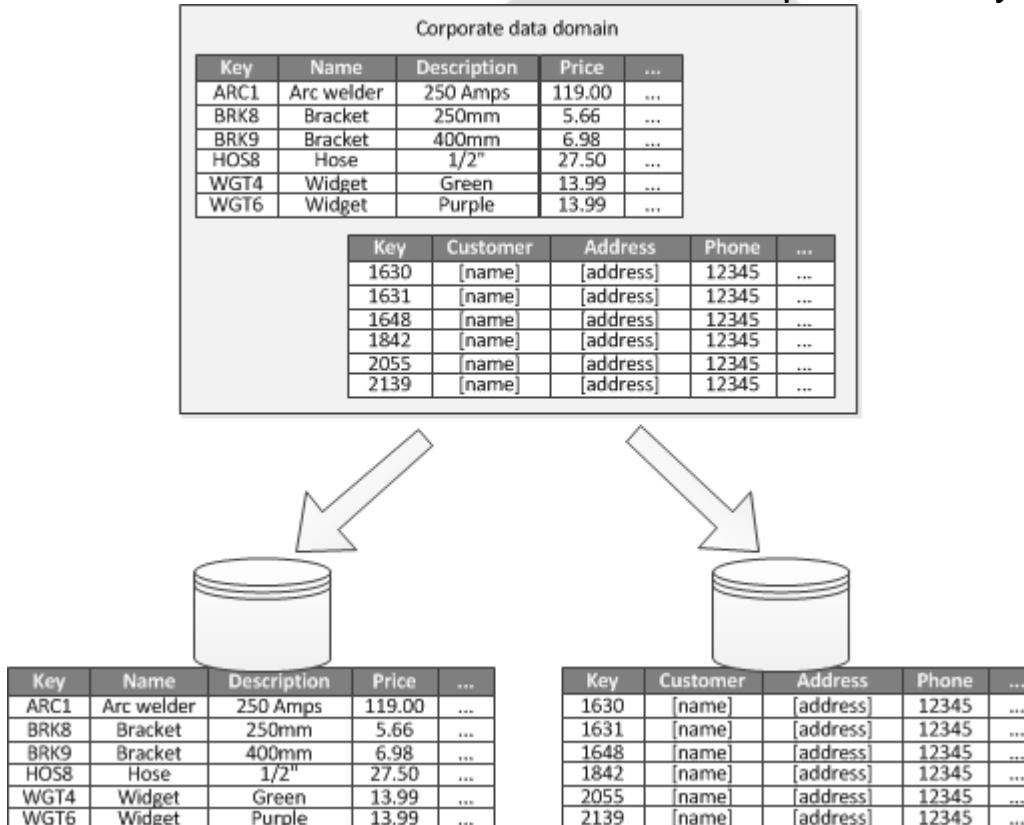
Vertical partitioning (2)

Advantages

- The most common use is to reduce the I/O and performance costs associated with fetching items that are frequently accessed
- Vertical partitioning operates at the entity level within a data store,
 - partially normalizing an entity to break it down from a wide item to a set of narrow items
 - ideally suited for column-oriented data stores such as HBase and Cassandra
- Relatively slow-moving data (product name, description, and price) can be separated from the more dynamic data (stock level and last ordered date)
 - Slow moving data is a good candidate for an application to cache in memory
- Sensitive data can be stored in a separate partition with additional security controls
- Vertical partitioning can reduce the amount of concurrent access that's needed

Functional partitioning

- Functional partitioning is a way to improve isolation and data access performance
 - suitable when possible to identify a bounded context for each distinct business area in an application
 - can be used to separate read-write data from read-only data
 - can help reduce data access contention across different parts of a system



Functionally partitioning data by bounded context or subdomain.

Rebalancing partitions

- As a system matures, need to adjust the partitioning scheme!
- For example,
 - individual partitions might start getting a disproportionate volume of traffic and become hot, leading to excessive contention
 - might have underestimated the volume of data in some partitions, causing some partitions to approach capacity limits
- Some data stores, such as Azure Cosmos DB, can automatically rebalance partitions
- In other cases, rebalancing is an administrative task that consists of two stages:
 - Determine a new partitioning strategy.
 - Which partitions need to be split (or possibly combined)?
 - What is the new partition key?
 - Migrate data from the old partitioning scheme to the new set of partitions.
- Migrations
 - Depending on the data store, might be able to migrate data between partitions while they are in use - online migration
 - If that's not possible, might need to make partitions unavailable while the data is relocated - offline migration

Rebalancing partitions

Migrations

- Offline migration
 - is typically simpler because it reduces the chances of contention occurring
 - Works as follows:
 - Mark the partition offline
 - Split-merge and move the data to the new partitions
 - Verify the data
 - Bring the new partitions online
 - Remove the old partition
 - Optionally, you can mark a partition as read-only in step 1, so that applications can still read the data while it is being moved
- Online migration
 - more complex to perform but less disruptive
 - similar to offline migration, except the original partition is not marked offline
 - Depending on the granularity of the migration process, the data access code in the client applications might have to handle reading and writing data that's held in two locations, the original partition and the new partition



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Data Partitioning in ML

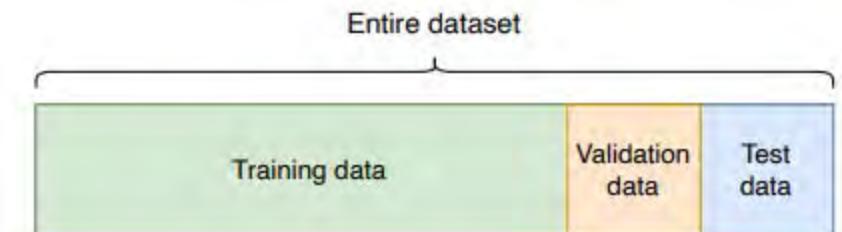
Pravin Y Pawar

Adapted from Machine Learning Engineering
By Andriy Burkov

Data Partitioning

Use three disjoint sets of examples: training set, validation set, and test set

- The **training set** is used by the machine learning **algorithm** to train the model
- The **validation set** is needed to **find the best values for the hyper parameters** of the machine learning pipeline
 - Data Scientists tries different combinations of hyper parameter values one by one, trains a model by using each combination, and notes the model performance on the validation set
 - hyper parameters that maximize the model performance are then used to train the model for production
- The **test set** is used for **reporting**
 - once you have your best model, you **test its performance on the test set and report the results**
- **Validation and test sets** are often referred to as **holdout sets**
 - contain the examples that the learning algorithm is not allowed to see



Data Partitioning(2)

Partitioning conditions

- To obtain good partitions of entire dataset into these three disjoint sets, as schematically partitioning has to satisfy several conditions
- Condition 1: Split was applied to raw data
 - Once access is given to raw examples, and before everything else, do the split - will allow avoiding data leakage
- Condition 2: Data was randomized before the split
 - Randomly shuffle examples first, then do the split
- Condition 3: Validation and test sets follow the same distribution
 - The examples in the test set are best representatives of the production data
 - Hence the need for the validation and test sets to follow the same distribution
- Condition 4: Leakage during the split was avoided
 - Data leakage can happen even during the data partitioning

Data Partitioning(3)

Splits

- There is no ideal ratio for the split!
- Size of data
 - In older literature (pre-big data), might find the recommended splits of either 70%/15%/15% or 80%/10%/10%
 - for training, validation, and test sets, respectively, in proportion to the entire dataset
 - Today, in the era of the Internet and cheap labor (e.g., Mechanical Turk or crowdsourcing), organizations, scientists, and even enthusiasts at home can get access to millions of training examples
 - makes it wasteful only to use 70% or 80% of the available data for training
- Reliability
 - The validation and test data are only used to calculate statistics reflecting the performance of the model
 - just need to be large enough to provide reliable statistics
 - How much is debatable
 - As a rule of thumb, having a dozen examples per class is a desirable minimum
 - If one can have a hundred examples per class in each of the two holdout sets
 - have a solid setup and the statistics calculated based on such sets are reliable

Data Partitioning(4)

Splits

- Model Architecture
 - The percentage of the split can also be dependent on the **chosen machine learning algorithm or model**
 - **Deep learning models** tend to significantly improve when exposed to **more training data**
 - **less true for shallow algorithms and models**
- Small size datasets
 - Proportions may depend on the size of the dataset
 - A small dataset **of less than a thousand examples** would do best with 90% of the data used for training
 - might decide to not have a distinct validation set, and instead simulate with the **cross-validation** technique

Data Partitioning(5)

Leakage During Partitioning

- Group leakage may occur during partitioning
- Imagine one have magnetic resonance images of the brains of multiple patients
 - Each image is labeled with certain brain disease, and the same patient may be represented by several images taken at different times
 - If one applies the partitioning technique (shuffle, then split), images of the same patient might appear in both the training and holdout data
 - The model might learn from the particularities of the patient rather than the disease
 - The model would remember that patient A's brain has specific brain convolutions, and if they have a specific disease in the training data, the model successfully predicts this disease in the validation data by recognizing patient A from just the brain convolutions
- The solution to group leakage is group partitioning
 - consists of keeping all patient examples together in one set: either training or holdout
 - Once again, highlights how important it is for the data analyst to know as much as possible about the data



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Data Versioning

Pravin Y Pawar

Adapted from AI Multitude blog

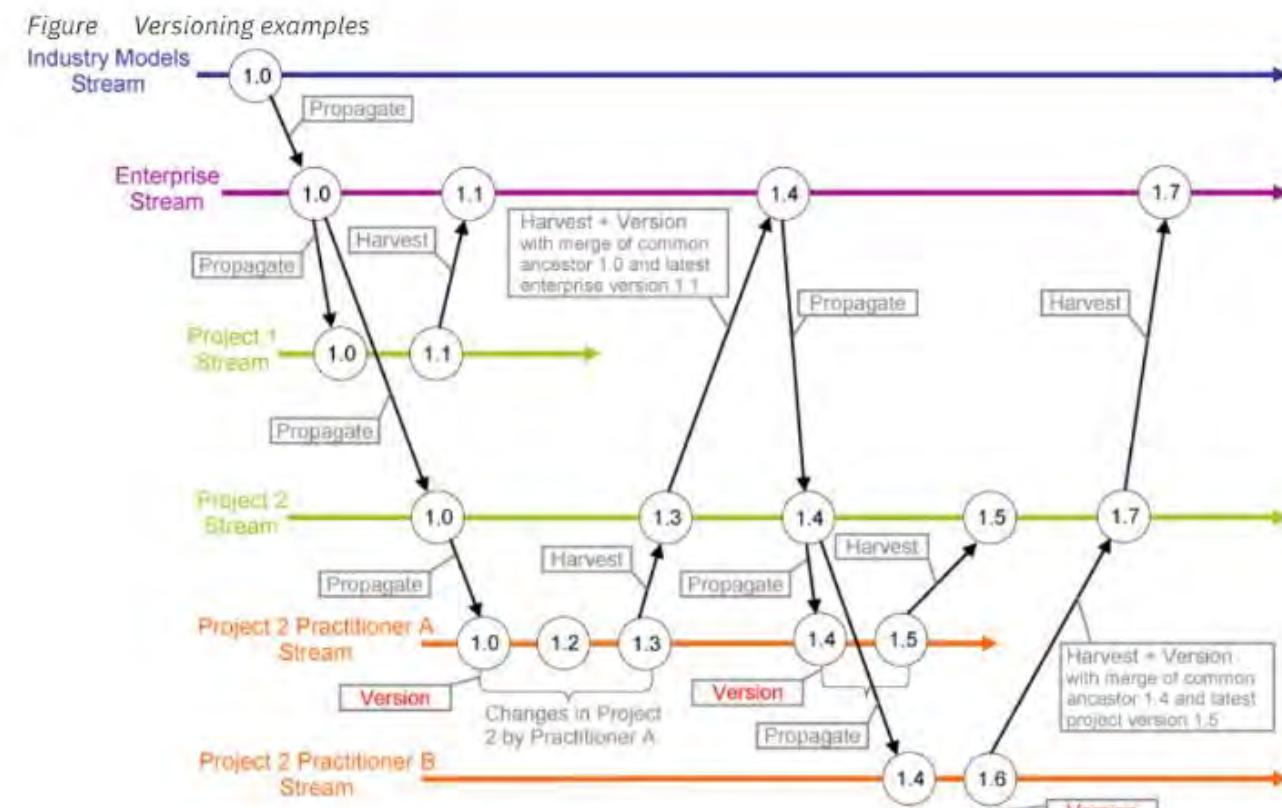
Data Versioning

Data Versioning

- 
- Riskless Testing
 - Useful for Measuring Business Performance
 - Ease Harmonization with Data Laws & Audit Process

Data Versioning(2)

- Data versioning is the storage of different versions of data that were created or changed at specific points in times
- Many different reasons for making changes to the data
 - Data scientists might test the ML models to increase efficiency
 - therefore make certain changes to the dataset
 - Datasets can also change over time due to the flow of information
 - storing older versions of data can help organizations replicate the previous environment



Source: [IBM](#)

Benefits of data versioning

Old versions of data is saved and kept at the company's disposal

- Preserving the working version while testing
 - AI/ML models work with the goal of maximizing business efficiency - normal for development teams to test new ways to increase efficiency
 - Introducing a new dataset into systems could be one of those exercises
 - However, in search of the uncertain better, no one wants to risk the previous working version
 - most of the engineers' attempts end up as inefficient trials - need to save the previous dataset
 - If an attempt fails, they simply reload the old working data set into the pipeline, preventing potential loss of business
- Measuring the business performance
 - Without the intervention of engineers, datasets can change over time - Sales data is changed by each transaction
 - Storing sales data from different years over a period of time can be insightful for businesses to understand consumer preferences
 - Consequently, versioning data can lead to a more profitable business by analyzing changes in customer trends
- Compliance and auditing benefits
 - Data versioning can help with both internal and external audits and compliance processes by ensuring data is stored from specific times
 - Some data protection regulations, such as GDPR, force companies to store certain data sources
 - Data versioning can save companies' time in meeting such requirements
 - can also be easier for companies to detect fraud if they have versioned their data

Main formats for data versioning

- There is no standard model for data versioning, but there are some common formats that are widely used:
- **The three-part semantic version number convention**
 - is the most common format for indicating different versions
 - For example, 3.2.4 indicates a specific data version
 - The left-hand side number (3) indicates a significant change between data versions
 - The middle number (2) indicates new features in a compatible manner
 - and the right-hand side number (4) indicates minor bug fixes compared to older versions
- **Naming data versions depending on their status**
 - For example, a dataset could be incomplete-complete, filtered-unfiltered, cleaned-uncleaned, etc.
 - Specifying this information could be helpful for practitioners, especially when they work together on a dataset via a cloud system
- **Naming data versions depending upon process**
 - Data version can be named subject to the latest process it is exposed to
 - For instance, normalized or adjusted according to something etc.

Options for versioning the data

Use file versioning

- File versioning
 - Manually saving versions to computer is one of the options for data versioning
- Appropriate for:
 - Small firms
 - Small firms with only a few data engineers or scientists working in the same location
 - Protecting sensitive information
 - If data contains particularly sensitive information, it should be viewed and interpreted only by a few executives and data engineers
 - Individual work
 - If the task is not suitable for teamwork, where different people cannot work together to achieve a final goal

Options for versioning the data(2)

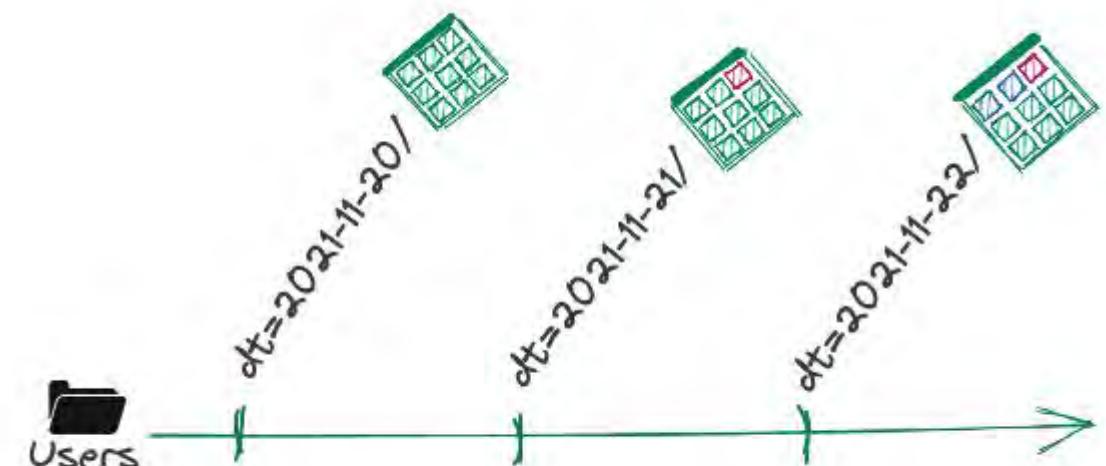
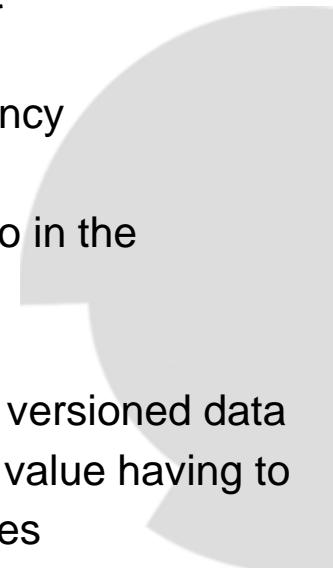
Using a data versioning tool

- Data versioning tool
 - Specialized tools offer an alternative to file versioning
 - can either develop own software or outsource it - providers offering such services, such as DVC, Delta Lake, and Pachyderm
- Data versioning tools are more suitable for companies that need:
 - Real-time editing
 - If more than one person is working on a dataset, using a specialized tool is more efficient
 - because file versioning does not allow real-time editing with a group of people
 - Collaboration from different locations
 - When people need to collaborate in different locations, using a software is more efficient than file versioning
 - Accountability
 - Data versioning software makes it possible to determine in which steps errors occur and who makes the error.
 - Consequently, the accountability of the team is enhanced

Data Versioning Implementation

Versioning Approach #1: Full Duplication

- One option is to **save a full copy** of it under a new location each time want a version of it
 - works best for **smaller datasets** with something like a daily versioning frequency
 - does create versioned data, it does so in the **least space-efficient way**
 - **code or queries** that interact with this versioned data will be **error-prone** with the correct date value having to be manually hardcoded in different places
 - Although not the most elegant solution, it is **an easy way to get started** versioning data



Versioning via saving a full copy of an example users dataset daily.

Data Versioning Implementation (2)

Versioning Approach #2: "Valid_from/to" Metadata

- A more space-efficient and incremental approach to versioning works by adding and maintaining two metadata fields in a tabular dataset
 - often named `valid_from` and `valid_to`
 - When updating a record in this dataset, make sure to never overwrite an existing record
 - Instead, append new records and update the `valid_to` field to the current timestamp for any record that would have been overwritten
- This approach works quite well for “time traveling” throughout a single collection of tabular data
- However, it provides only one method of interacting with the versions—which is to add filters to queries on the metadata fields

Orders:			
order_id	status	valid_from	valid_to
1	pending		2021-10-16
1	shipped	2021-10-16	2021-10-18
1	delivered	2021-10-18	



```
Select * from orders  
Where valid_from ≤ '2021-10-17'  
and valid_to ≥ '2021-10-17'
```

Results:			
order_id	status	valid_from	valid_to
1	shipped	2021-10-16	2021-10-18



Using query filters to get the state of the Orders table on Oct. 17.

Data Versioning Implementation (3)

Versioning Approach #3: First-class Data Version Control

- First two approaches can be summarized as “**Let me add a bit of versioning to the data I already have**”
 - Instead, should think of versioning as a first-class citizen of data environment
 - An inherent property of any data we introduce into the system
- To make this possible - need to solve a few challenges.
 - Minimize the storage footprint of data versioning
 - means not creating copies of data objects that remain unchanged between versions
 - Expose operations that let us interact directly with the versions
 - Things like “create a version”, “delete a version”, “compare two versions” are examples
 - Work the same over any scale of data, data format, and both structured and unstructured data
- Now, these are not simple problems
 - And probably won’t hack way to a solution in an afternoon or even a weekend
 - One of the more popular approaches to solving these extends the git version control model to data
 - projects like lakeFS, DVC, and git LFS

Challenges to data versioning

- Limited storage space
 - Each versioning of data means that more storage space is required
 - For companies that produce or use large amounts of data, it would therefore be costly to version the data too often
 - important for companies to find an optimal balance between the benefits of versioning and the costs incurred by storage
- Security issues
 - Ensuring data security is essential for businesses to protect their reputation
 - However, as more and more versions of data are stored, the risk of data loss or leakage increases
 - For cloud users, in particular, this risk is even greater as they simply outsource their IT functions, giving them less control over their data
 - important for organizations to assess and understand this risk in order to determine an optimal data versioning strategy
- Choosing the right provider
 - If decide to use a data versioning tool, want to choose the most suitable option that meets business requirements
 - Different cloud providers offer different features and charge different prices
 - advisable to evaluate the different options in order to ensure cloud cost optimization
 - Should compare the tools according to the following criteria:
 - Open source or not
 - Storage capacity
 - Has a user friendly interface or not
 - Support of most common clouds (like AWS) and storage types or not
 - Cost



Thank You!

In our next session:

[Q&A with Data Scientists from Theta Tech AI: MLOps for clinical research studies \[Register now\]](#)

Table of contents

Best 7 Data Version Control Tools That Improve Your Workflow With Machine Learning Projects

5 min

 Jakub Czakon

 14th November, 2022

Machine Learning Tools

Keeping track of all the data you use for models and experiments is not exactly a piece of cake. It takes a lot of time and is more than just managing and tracking files. You need to ensure everybody's on the same page and follows changes simultaneously to keep track of the latest version.

You can do that with no effort by using the right software! **A good data version control tool will allow you to have unified data sets with a strong repository of all your experiments.**

It will also enable smooth collaboration between all team members so everyone can follow changes in real-time and always know what's happening.

It's a great way to systematize data version control, improve workflow, and minimize the risk of

Table of contents

So check out these top tools for data version control that can help you automate work and optimize processes.

Data versioning tools are critical to your workflow if you care about reproducibility, traceability, and ML model lineage.

They help you get a version of an artifact, a hash of the dataset or model that you can use to identify and compare it later. Often you'd log this data version into your metadata management solution to make sure your model training is versioned and reproducible.

How to choose a data versioning tool?

To choose a suitable data versioning tool for your workflow, you should check:

- **Support for your data modality:** how does it support video/audio? Does it provide some preview for tabular data?
- **Ease of use:** how easy is it to use in your workflow? How much overhead does it add to your execution?
- **Diff and compare:** Can you compare datasets? Can you see the diff for your image directory?
- **How well does it work with your stack:** Can you easily connect to your infrastructure, platform, or model training workflow?

- **Can you get your team on board:** If your team does not adopt it, it doesn't matter how good the tool is. So keep your teammates skillset in mind and preferences in mind.

Here're are a few tools worth exploring.

Table of contents

Best data version control tools

1. Neptune

Search or filter runs

(A) Tags all of showcase-run, group-by-dataset X		RUN LABEL					Recent searches ▾	
(B) datasets/train	A	Id	test_score	max_depth	n_estimators	datasets/train	datasets/test	
532c60d9154cff903ef6 Displaying top 6 of 6 runs Show all	<input type="checkbox"/>	DAT-36	0.92	3	5	532c60d9154cff90...	92ad1240c7f8bd5ef...	
	<input type="checkbox"/>	DAT-39	0.94	3	6	532c60d9154cff90...	92ad1240c7f8bd5ef...	
	<input type="checkbox"/>	DAT-37	0.94	2	6	532c60d9154cff90...	92ad1240c7f8bd5ef...	
	<input type="checkbox"/>	DAT-35	0.94	2	6	532c60d9154cff90...	92ad1240c7f8bd5ef...	
	<input type="checkbox"/>	DAT-40	0.98	2	7	532c60d9154cff90...	92ad1240c7f8bd5ef...	
	<input type="checkbox"/>	DAT-38	0.98	2	5	532c60d9154cff90...	92ad1240c7f8bd5ef...	
0aa0ba4dae14431a124	<input type="checkbox"/>	DAT-27	0.94	2	9	0aa0ba4dae14431...	92ad1240c7f8bd5ef...	

Example dashboard in Neptune

Neptune is an ML metadata store that was built for research and production teams that run many experiments.

You can log and display pretty much any ML metadata from hyperparameters and metrics to videos,

Table of contents

Neptune artifacts let you version datasets, models, and other files from your local filesystem or any S3-compatible storage with a single line of code. Specifically, it saves:

- **Version** (hash) for the file or folder
- **Location** of the file or folder
- Folder **structure** (recursively)
- **Size** of the file or folder

Once logged, you can use Neptune UI to group runs on dataset versions or see how the artifacts changed between runs.

When it comes to data versioning, Neptune is a very lightweight solution, and you can get going quickly. That said, it may not give you everything you need data-versioning-wise.

On the other hand, you get [experiment tracking](#) and [model registry](#) all in one place and use a [flexible metadata structure](#) to organize training and production metadata the way you want to. It is like a dictionary or a folder structure that you create in code and display in the UI.

If you are wondering if it will fit your workflow:

- check out [case studies](#) of how people set up their MLOps tool stack with Neptune
- explore an [example public project](#) about dataset versioning
- run a [hello world](#) or [dataset versioning example](#) in Colab and see for yourself

- but if you are like me, you would like to compare it to other tools in the space like DVC, Pachyderm, or wandb. So here are many deeper feature-by-feature comparisons to make the evaluation easier.

Table of contents

2. Pachyderm

The screenshot shows the Pachyderm dashboard interface. At the top, there's a search bar labeled "Search Pachyderm" with a magnifying glass icon and an "X" button. Below the search bar, the word "inference" is displayed in large blue letters, with a small circular icon containing a green arrow pointing right next to it. To the right of "inference", the text "Last updated a few seconds ago" is shown. On the left side, there's a sidebar with a "PACH DASH" header and a "Home" button. Below the header, there are three main navigation items: "Repos" (represented by a list icon), "Pipelines" (represented by a pipeline icon), and "Jobs" (represented by a briefcase icon). The "Pipelines" item is currently selected, as indicated by its bolded text. The main content area is titled "Table of contents". It displays information about the "inference" pipeline, including its status as a "Computed Output Repo" last updated a few seconds ago, and its metrics: 312 files, 0 dirs, 763820 B, and 314 commits. Below this, there are two columns of metrics: "1 active jobs" (with a monitor icon) and "314 output commits" (with a diamond icon); "2 Inputs" (with a diamond icon) and "1 version" (with an upward arrow icon); and "793.4 KB generated" (with a circle icon) and "278ms avg runtime" (with a briefcase icon). Further down, there's a section titled "Takes input from" which lists two repos: "attributes" (Manually Ingested Repo, 0 files, 8144 B) and "model" (Computed Output Repo, 0 files, 5322 B). An "See all details..." link is also present.

Pachyderm is a complete version-controlled data science platform that helps to control an end-to-end machine learning life cycle. It comes in three different versions, Community Edition (open-

source, with the ability to be deployed anywhere), Enterprise Edition (complete version-controlled platform), and Hub Edition (a hosted version, still in beta).

It's a great platform for **flexible collaboration** on any kind of machine learning project.

Table of contents

- Pachyderm lets you continuously update data in the master branch of your repo, while experimenting with specific data commits in a separate branch or branches
- It supports any type, size, and number of files including binary and plain text files
- Pachyderm commits are centralized and transactional
- Provenance enables teams to build on each other work, share, transform, and update datasets while automatically maintaining a complete audit trail so that all results are reproducible

Check also

[The Best Pachyderm Alternatives](#)

3. DVC

The screenshot shows the DVC website's homepage. At the top, there's a navigation bar with links for FEATURES, DOC, BLOG, CHAT, GITHUB, SUPPORT, and a prominent blue "Get Started" button. Below the navigation is a large "Table of contents" section. Three main features are highlighted with icons and descriptions:

- ML project version control**: An icon showing a flowchart with nodes labeled "Cloud" and "Local Cache".
- ML experiment management**: An icon showing a network graph with colored nodes.
- Deployment & Collaboration**: An icon showing a central node connected to several smaller nodes, with a red gear icon and the text "Deploy to production".

DVC is an open-source version control system for machine learning projects. It's a tool that lets you define your pipeline regardless of the language you use.

When you find a problem in a previous version of your ML model, DVC saves your time by leveraging code data, and pipeline versioning, to give you reproducibility. You can also train your model and share it with your teammates via DVC pipelines.

DVC can cope with versioning and organization of big amounts of data and store them in a well-organized, accessible way. It focuses on data and pipeline versioning and management but also has some (limited) experiment tracking functionalities.

DVC – summary:

- Possibility to use different types of storage— it's storage agnostic
- Full code and data provenance help to track the complete evolution of every ML model
- Reproducibility by consistently maintaining a combination of input data, configuration, and the

Table of contents

- A built-in way to connect ML steps into a DAG and run the full pipeline end-to-end
- Tracking failed attempts
- Runs on top of any Git repository and is compatible with any standard Git server or provider

See also

[DVC vs Neptune comparison](#)

4. Git LFS

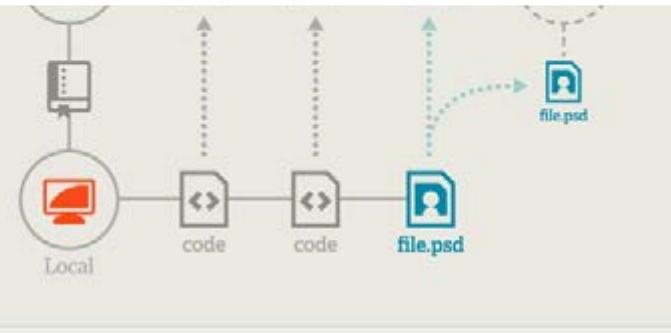


Git Large File Storage

[Docs](#) [Downloads](#) [Source](#)

Table of contents

Git Large File Storage (LFS) replaces large files such as audio samples, videos, datasets, and graphics with text pointers inside Git, while storing the file contents on a remote server like GitHub.com or GitHub Enterprise.

 [Download v2.12.1 \(Windows\)](#)

Git Large File Storage (LFS) is an open-source project. It **replaces large files** such as audio samples, videos, datasets, and graphics with text pointers inside Git, while storing the file contents on a remote server like GitHub.com or GitHub Enterprise.

It allows you to **version large files**—even those as large as a couple GB in size—with **Git, host more** in your Git repositories with external storage, and to **faster clone and fetch** from repositories that deal with large files.

At the same time, you can keep your workflow and the same access controls and permissions for large files as the rest of your Git repository when working with a remote host like GitHub.



The screenshot shows the DoltHub interface for a GitHub repository named 'ip-to-country'. The repository has a single branch 'master'. The main area displays a table titled 'IPv4ToCountry' with columns: IPFrom (Primary key int), IpTo (Primary key int), Registry (String), AssignedDate (Int), CountryCode2Letter (String), CountryCode3Letter (String), and Country (String). Below the table is a 'SQL Console' window containing the following SQL code:

```

1 SELECT *
2 FROM 'IPv4ToCountry'
3 LIMIT 200;
4
5
6
7
8
9
10

```

	IPFrom	IpTo	Registry	AssignedDate	CountryCode2Letter	CountryCode3Letter	Country	AU	AUS
16777216	16777471	apnic	1313020800					AU	AUS
16777472	16777727	apnic	1302739200					CHN	
16777728	16778239	apnic	1302739200					CHN	
16778240	16779263	apnic	1302566400					AU	AUS
16779264	16781311	apnic	1302566400					CHN	
16781312	16785407	apnic	1302566400					JP	JPN
16785408	16793599	apnic	1302566400					CHN	
16793600	16809983	apnic	1302566400					JP	JPN

Dolt is a SQL database that you can *fork*, *clone*, *branch*, *merge*, *push*, and *pull* just like a git repository. Dolt allows data and schema to evolve together to **make a version control database a better experience**. It's a great tool to collaborate on with your team.

You can freely connect to Dolt just like to any MySQL database to run queries or update the data using SQL commands.

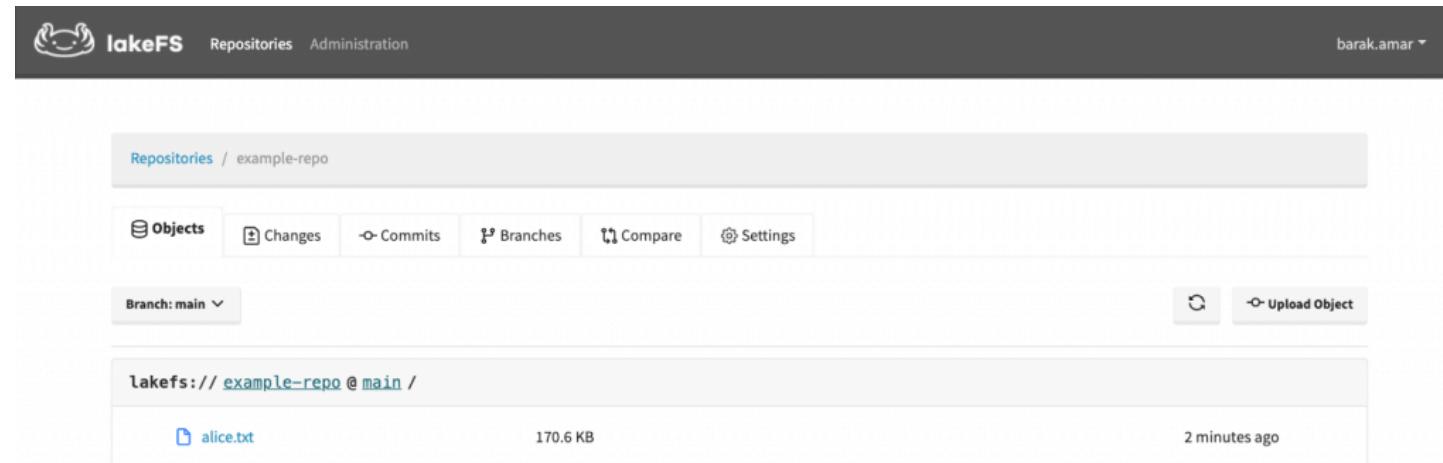
Use the command line interface to import CSV files, commit your changes, push them to a remote, or merge your teammate's changes.

All the commands you know for Git work exactly the same for Dolt. Git versions files, Dolt versions tables.

There's also **DoltHub** – a place to share Dolt databases.

Table of contents

6. lakeFS



The screenshot shows the lakeFS web interface. At the top, there is a dark header bar with the lakeFS logo, navigation links for 'Repositories' and 'Administration', and a user dropdown for 'barak.amar'. Below the header, the main interface has a light gray background. It displays the path 'Repositories / example-repo'. A navigation bar below the path includes tabs for 'Objects' (selected), 'Changes', 'Commits', 'Branches', 'Compare', and 'Settings'. A dropdown for 'Branch: main' is shown. On the right side of the navigation bar are refresh and upload buttons. The main content area shows a list of objects under the heading 'lakefs:// example-repo @ main /'. The first item listed is 'alice.txt', which is 170.6 KB in size and was uploaded 2 minutes ago. There is a small icon next to the file name.

lakeFS is an open-source platform that provides a Git-like branching and committing model that scales to Petabytes of data by utilizing S3 or GCS for storage.

This branching model makes your data lake ACID-compliant by allowing changes to happen in isolated branches that can be created, merged, and rolled back atomically and instantly.

lakeFS has three main areas that let you focus on different aspect of your ML models:

1. **Development Environment for Data:** has tools that you can use to isolate snapshot of the lake you can experiment with while others are not exposed; reproducibility to compare changes and

Table of contents

3. **Continuous Data Deployment:** ability to quickly revert changes to data; providing consistency in your datasets; testing of production data to avoid cascading quality issues

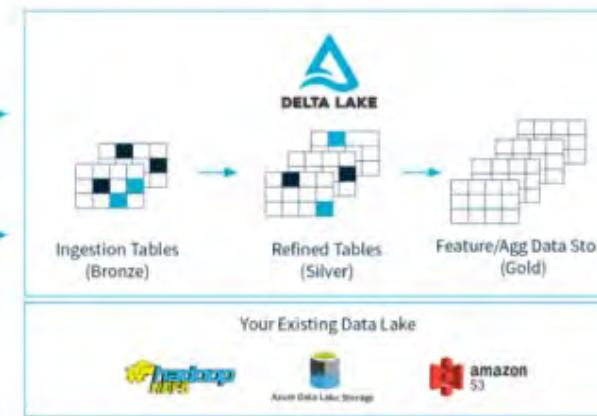
lakeFS is a great tool for focusing on a specific area of your datasets to make ML experiments more consistent.

7. Delta Lake

The screenshot shows the official Delta Lake website. At the top, there's a dark header with the "DELTA LAKE" logo on the left and navigation links for "DOCS", "COMMUNITY", "CODE", "RELEASES", and "RESOURCES". On the far right, there are icons for GitHub and Twitter. Below the header is a banner with the text "Read the VLDB paper - Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores".

Table of contents

Delta Lake is an open-source storage layer that brings ACID transactions to Apache Spark™ and big data workloads.



Latest News

[Salesforce Engineering | Delta Lake Blog Series](#)
October 20, 2020

[Getting Started with Delta Lake](#)
September 16, 2020

[Delta Lake Sessions at Spark+AI Summit North America 2020](#)
June 23, 2020

[Delta Lake 0.7.0 Released](#)
June 16, 2020

Delta Lake is an open-source storage layer that brings **reliability to data lakes**. Delta Lake provides ACID transactions, scalable metadata handling, and unifies streaming and batch data processing. It runs on top of your existing data lake and is fully compatible with Apache Spark APIs.

Delta Lake – summary:

- **Scalable metadata handling:** Leverages Spark's distributed processing power to handle all the metadata for petabyte-scale tables with billions of files at ease.

- **Streaming and batch unification:** A table in Delta Lake is a batch table as well as a streaming source and sink. Streaming data ingest, batch historic backfill, interactive queries all just work out of the box.
- **Schema enforcement:** Automatically handles schema variations to prevent insertion of bad

Table of contents

- **Data versioning enables rollbacks, full historical audit trails, and reproducible machine learning experiments**
- **Supports merge, update, and delete operations** to enable complex use cases like change-data-capture, slowly-changing-dimension (SCD) operations, streaming upserts, and so on.

To wrap it up

Now that you have the list of the best tools for data versioning, you “just” need to figure out how to make it work for you and your team.

That can be tricky.

Some things to consider when choosing a data versioning are:

- **How easy is it to set up:** You may not have the time, needs, or budget to test something heavy right now.
- **Can you get your team onboard:** Sometimes, the solution is great, but you need more software engineering-oriented mindset to use it. Some ML researchers or data scientists may not end up using it.
- **What tool stack are you using today:** Are you using specific tools, infrastructure, or platform that has good integration with a particular data versioning solution. In that case, probably the best

option is to just go with that.

- **Data modality:** Is it images, tables, text, all? Sometimes the tool doesn't support your modality very well as it was built with a different use case in mind.

Table of contents

Reach out to me, and let's see what I can do!

Jakub Czakon

Mostly an ML person. Building MLOps tools, writing technical stuff, experimenting with ideas at Neptune.

Follow me on

Read next

Best AI & ML Tools When You Work With Projects for

Table of contents

Telecom companies have a lot of business and functional divisions that make them tick.

Data scientists that work in telecom can have various tasks to take care of depending on the division. There could be a data science team that improves customer experience, or one that powers the product & engineering division.

In this article, we'll look at popular use cases of data science and related tools in a telecom company, from my perspective as an ex-telecom data scientist.

My typical day as a Data Scientist in telecom

As a data scientist in the customer experience team, there are three main types of tasks that my role involved.

Business As Usual (BAU)

BAU is where you track certain KPIs, or tune/refresh an existing machine learning model.

Honestly, it's the least interesting work for a passionate data scientist, since it involves very little innovation and plenty of redundant tasks.

But it's a very important part of telecom companies, because it helps leadership and executives

[Continue reading](#)

Table of contents

**Building Visual Search Engines with
Kuba Cieślik**

by **Stephen Oladele**, 17 min read

[Read more](#)

**Deploying ML Models on GPU With
Kyle Morris**

by **Stephen Oladele**, 25 min read

[Read more](#)

Table of contents

ML Collaboration: Best Practices From 4 ML Teams

by Vidhi Chugh, 7 min read

[Read more](#)

Classification in ML: Lessons Learned From Building and Deploying a Large-Scale Model

by Shibsankar Das, 7 min read

[Read more](#)

Newsletter

Top MLOps articles, case studies, events (and more) in your inbox every month.

Your e-mail

[Get Newsletter](#)

PRODUCT

DOCUMENTATION

Table of contents

COMMUNITY

COMPANY

[The Best MLOps Tools](#)

[MLOps at a Reasonable Scale](#)

[ML Metadata Store](#)

[MLOps: What, Why, and How](#)

[Experiment Tracking in Machine Learning](#)

[Terms of service](#) [Privacy policy](#)

Copyright © 2022 Neptune Labs. All rights reserved.



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Test Data Management

Pravin Y Pawar

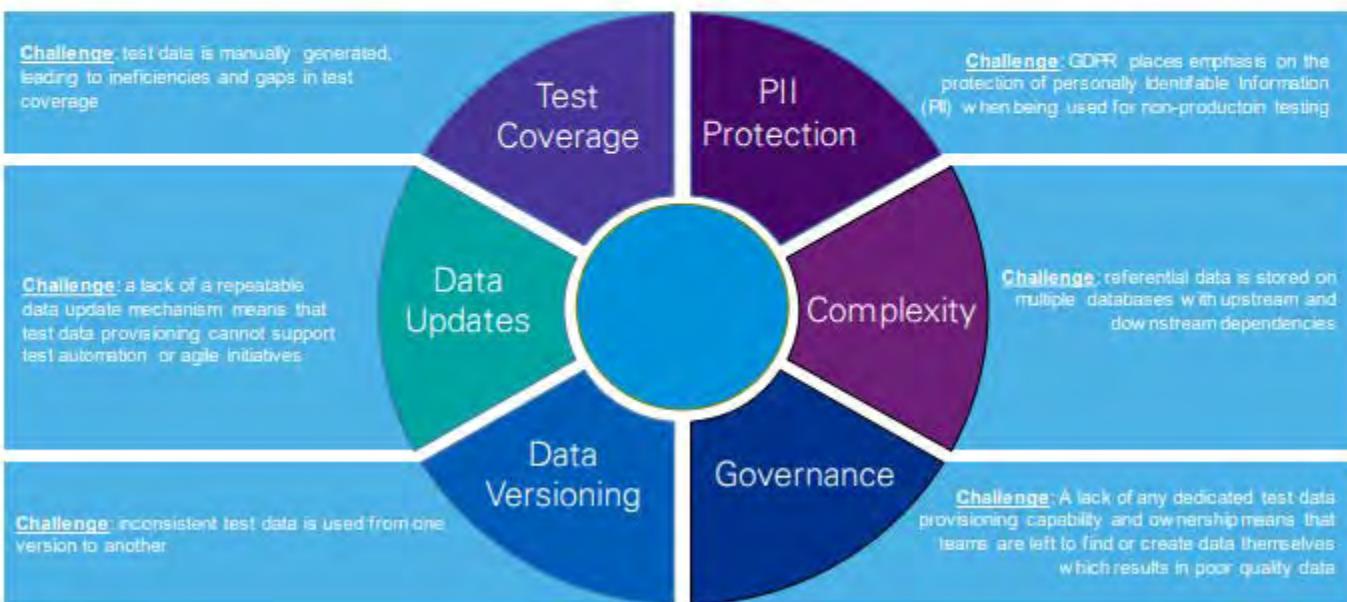
Adapted from
[KPMG Test data management challenges](#)

Test Data Management

Challenges

Test Data Management Challenges

High quality test data is a cornerstone of successful testing, but there are many challenges to effectively managing it. KPMG takes a structured approach to developing and implementing a test data management strategy that addresses these challenges to deliver quality test solutions.

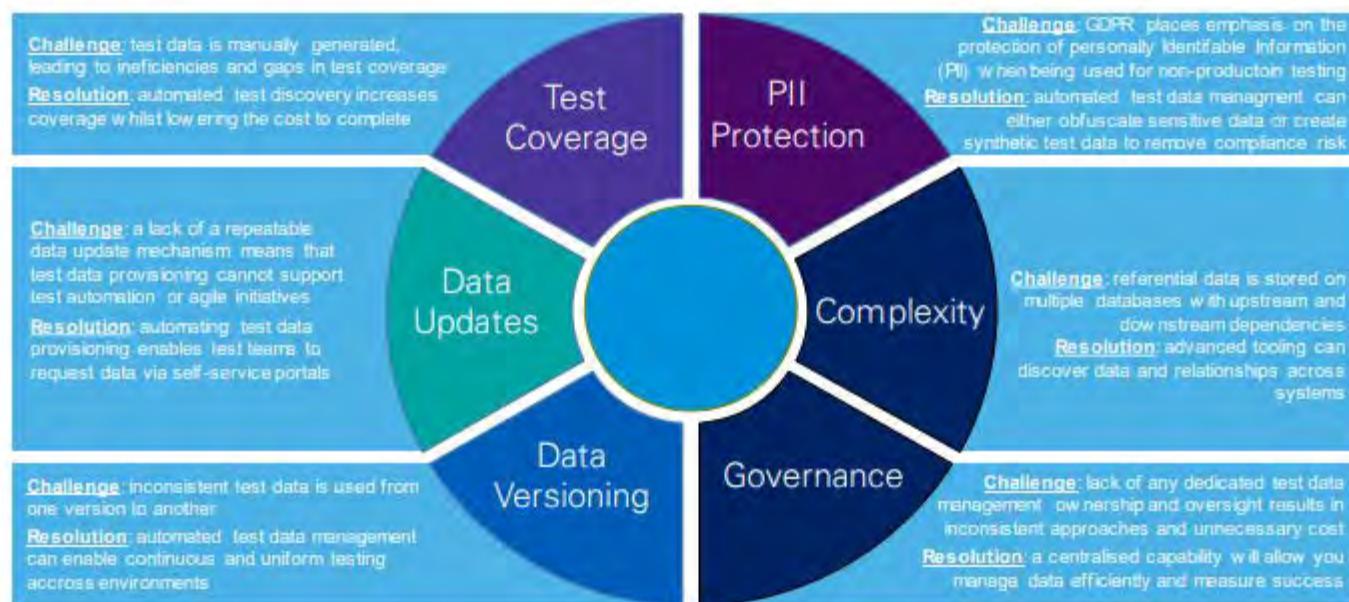


Test Data Management

Resolution

Solving The Test Data Management Challenges

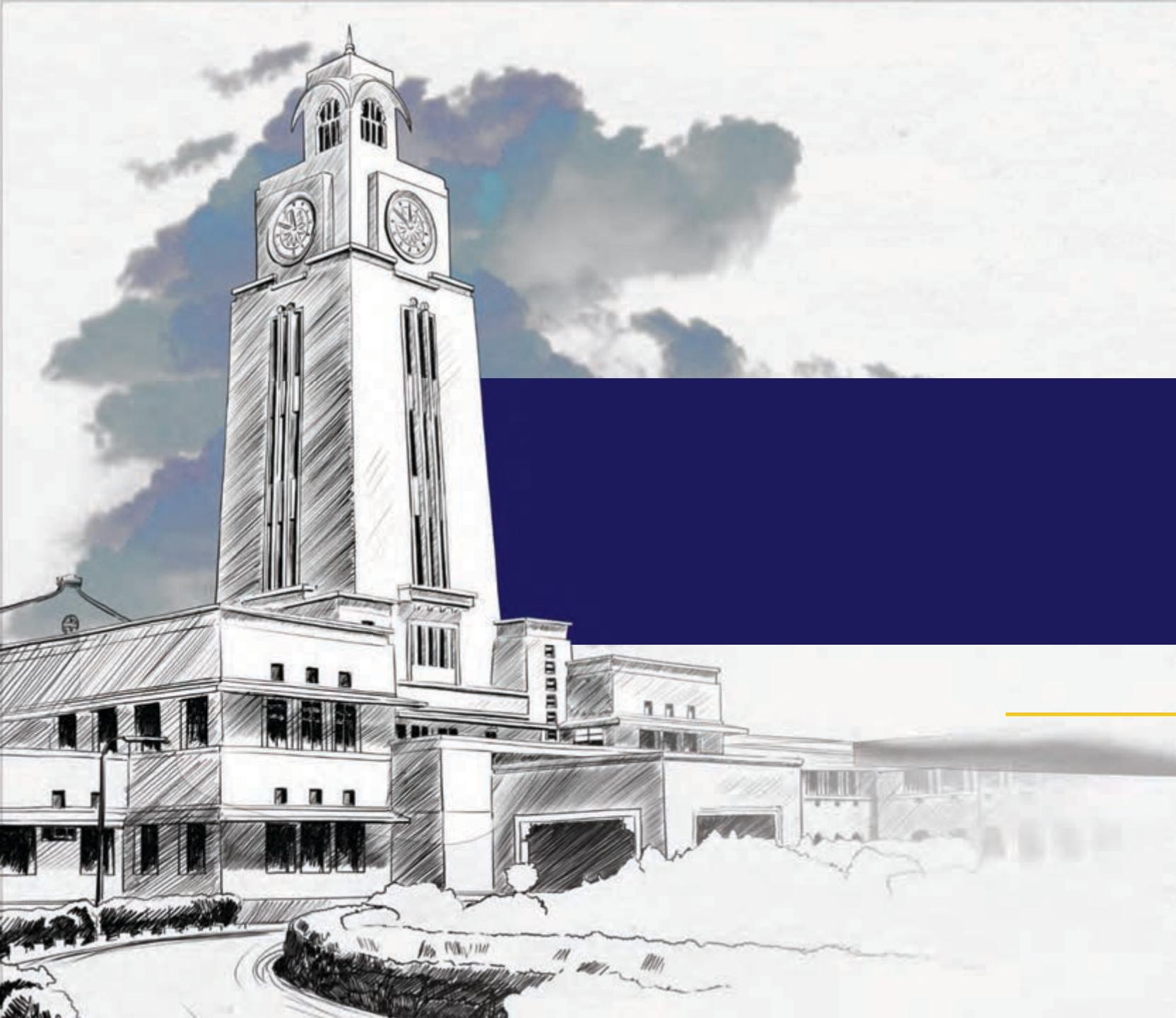
To successfully meet the challenges in managing test data in your organization a strategy must be underpinned with tools that automate your processes. With our deep technical knowledge and industry expertise, KMPG can help develop and implement the most appropriate strategy for your organisation.





Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Features

Pravin Y Pawar

Adapted from Reliable Machine Learning
by Cathy Chen

Features

Definitions

- The data is just the data. Features are what make it useful for ML!
 - A feature is any aspect of the data that we determine is useful in accomplishing the goals of the model
 - “we” includes the humans building the model or many times can now include automatic feature-engineering systems
- A feature is a specific, measurable aspect of the data, or a function of it.
- Features are used to build the models
 - Structures that connects models back to data used to assemble the model
- Model is set of rules that takes data and uses it to generate predictions about world
 - true for model architecture and configured model
- Trained model is formula for combining a collection of features definitions used to extract feature values from real data
- More than just pieces of raw data, but rather involve some kind of preprocessing!

Features

Examples

- Concrete examples of features
 - From a web log, information about the customer (say, browser type).
 - Individual words or combinations of words from text a human enters into an application.
 - The set of all the pixels in an image, or a structured subset thereof.
 - Current weather at the customer's location when they loaded the page.
 - Any combination or transformation of features can itself be a feature.
- Typically, features contain smaller portions of structured data extracted from the underlying training data
 - As modeling techniques continue to develop, it seems likely that more features will start to resemble raw data rather than these extracted elements
- For example, a model training on text currently might train on words or combinations of words
 - but expect to see more training on paragraphs or even whole documents in the future
 - has significant implications for modeling

Feature Definition and Feature Values

- Feature definition
 - code (or an algorithm or other written description) that describes the information getting extracted from the underlying data
 - not any specific instance of that data being extracted
- For example, feature definition is
 - “Source country of the current customer obtained by taking the customer’s source IP address and looking it up in the geolocation data service” is a feature definition
 - Any particular country—for example, the “Dominican Republic” or “Russia” — would not be a feature definition
- Feature value
 - specific output of a feature definition applied to incoming data
 - “Dominican Republic” is a feature value that might be obtained by determining that the current customer’s source IP address

Feature Selection and Engineering

- Feature selection is process by which features are identified in data useful for models
- Feature engineering is process by which features are extracted and transformed into usable state
- Figure out what matters and what doesn't!
- Building and managing features has changed over years and will continue to evolve
 - Moving from entirely manual process to a mostly automated one to completely automated one
- Refer to processes of selecting and transforming features jointly as feature engineering

Human-driven Feature Engineering

- Process normally starts with human intuition based on an understanding or the domain of the problem
 - or at least detailed consultation with experts
- Understanding the underlying problem area is key, and more specificity is better
 - ML engineer spends time with a dataset and a problem
 - uses a set of statistical tools to evaluate the likelihood that a particular feature, or several features in combination with one another, will be useful in the task
- Next, ML engineers typically brainstorm to generate a list of possible features
 - Of these features, one of them is may be much more likely to be useful than the others
 - up to humans to generate and evaluate those ideas by using the ML platform and model metrics

Algorithmic / Automated Feature Engineering

- For algorithmic feature engineering, the process is considerably more automatic and data bound
 - AutoML is capable of not only selecting from identified features but also being able to programmatically apply common transforms (like log scaling or thresholding) to existing features
 - Embedding is one of other ways which enables to learn something (an embedding) about the data without explicitly specifying it
- Still, algorithms are generally able to identify only features that exist in the data
 - whereas humans can imagine new data that could be collected that might be relevant
- Even more importantly, humans understand the process of data collection, including any likely systemic bias
 - might have a material impact on the value of the data

Lifecycle of a Feature

- Feature definitions are created to fill a need, evaluated against that need, and eventually discarded as either the model is discarded or better features are found to accomplish the same goal
- Simplified version of the lifecycle of a feature (both the definition and the representative values):
 - Data collection/creation
 - Data cleaning/normalization/preprocessing
 - Candidate feature definition creation
 - Feature value extraction
 - Storage of feature values in a feature store
 - Feature definition evaluation
 - Model training and serving using feature values
 - (Usually) Update of feature definitions
 - (Sometimes) Deletion of feature values
 - (Eventually) Discontinuation of a feature definition

Lifecycle of a Feature(2)

- Data collection/creation
 - No data, no features!
 - need to collect or create data in order to create features
- Data cleaning/normalization/preprocessing involves coarser preprocessing steps:
 - eliminating obviously malformed examples,
 - scaling input samples to a common set of values,
 - possibly even deleting specific data that we should not train on for policy reasons
 - might seem outside the feature-engineering process, but no features can exist until the data exists and is in a usable form
- Candidate feature definition creation
 - Using either subject matter expertise plus human imagination or automated tools
 - develop a hypothesis for which elements or combinations of data are likely to accomplish the models goals

Lifecycle of a Feature(3)

- Feature value extraction
 - Need to write code that reads the input data and extracts the features from the data
 - In some simple situations, might need to do this inline as part of the training process
 - But if we expect to train on the same data more than a few times,
 - it's probably sensible to extract the feature from the raw data and store it for later efficient and consistent reading
- If application involves online serving
 - need a version of this code to extract the same features from the values available at serving
 - in order to use them to perform inference in model
 - Under ideal circumstances, the same code can extract features for training and for serving
 - but may have additional constraints in serving that are not present in training
- Storage of feature values in a feature store
 - place to put the features
 - is just a place to write extracted feature values so that they can be quickly and consistently read during training a model

Lifecycle of a Feature(4)

- Feature definition evaluation
 - Once features are extracted, most likely will build a model using them or add new features to an existing model in order to evaluate how well they work
 - looking for evidence that the features provide the value that we were hoping they would
- Evaluation of feature definitions comes in two distinct phases that are connected
 - First, need to determine whether the feature is useful at all - coarse-grained evaluation
 - simply trying to decide whether to continue working on integrating the feature into model
 - Next phase occurs if decide to keep that feature
 - need a process to continuously evaluate the quality and value (compared to cost) of the future to determine that it is still working the way we expect and providing the value we expect even several years from now
- Model training and serving using feature values
 - Perhaps this is obvious, but the entire point of having features is to use them to train models
 - then use the resulting models for a particular purpose

Lifecycle of a Feature(5)

- (Usually) Update of feature definitions
 - Frequently have to update the definition of a feature, either to fix a bug or simply to improve it in some way
 - If keep track of versions for feature definitions, this will be much easier
 - can update the version and then, optionally, reprocess older data to create a new set of feature values for the new version
- (Sometimes) Deletion of feature values
 - Sometimes need to be able to delete feature values from feature store
 - can be for policy/governance reasons;
 - can also be for quality/efficiency reasons- corrupted, sampled in a biased way or just too old to be useful
- (Eventually) Discontinuation of a feature definition
 - Either found a better way to provide the value that this feature definition provides
 - or find that the world has changed enough that this feature no longer provides any value
 - Eventually, will decide to retire the feature definition (and values) entirely
 - Need to
 - remove any serving code that refers to the feature,
 - remove the feature values in feature store,
 - cancel the code that extracts from the data
 - proceed to delete the feature code



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Feature Systems

Pravin Y Pawar

Adapted from Reliable Machine Learning
by Cathy Chen

Feature Systems

- Need to decompose the work into several subsystems to
 - successfully manage the flow of data through systems
 - turn data into features that are usable by training system and manageable by our modelers
- Let's walk through the feature systems starting with raw data and ending up with features stored in a format ready to be read by a training system
 - Data Ingestion system
 - Feature store
 - Feature quality evaluation system

Data ingestion system

- Have to write software that reads raw data, applies feature extraction code to that data, and stores the resulting feature values in the feature store
- In the case of one-time extraction, even for a very large amount of data
 - may be a relatively ad hoc process with code designed to be run once
- But in many cases, the process of extracting data is a separate production system all its own
- When have many users or repeated use of the data ingestion system
 - it should be structured as an on-demand, repeatable, monitored data processing pipeline
 - the biggest variable is user code
- Need a system whereby feature authors write code that identifies features and extracts them to store in the feature store
 - an either let feature authors run their code themselves, which imposes a substantial operational burden on them
 - can provide them a development engineering environment
 - helping them writing reliable feature-extraction code so that it can be run on that code reliably in data ingestion system

Data ingestion system(2)

- Need to build a few systems to facilitate feature authors writing reliable and correct features
- Versioning
 - should note that features should be versioned
 - likely want to substantially change a feature over time, perhaps because of changes in data that it merges with or other factors related to the data a feature version helps keep the transition clear and avoid unintended consequences of the change
- Test Systems
 - need a test system that checks feature-extraction code for basic correctness
 - need a staging environment for running proposed feature extraction on a certain number of examples
 - provide those to the feature author along with basic analysis tools to ensure that the feature is extracting what it is expected to extract
 - may want to allow the feature to be run or may want additional human review for reliability concerns
 - dependence on external data, for example
 - **The more work we do here, the more productive feature authors will be.**

Feature Store

- A storage system designed to store extracted feature (and label) values
 - so that they can be quickly and consistently read during training a model and even during inference
 - most useful, however, in larger, centrally managed services, especially when the features (definitions and values both) are shared among multiple models
- Recognizing the importance of putting feature and label data in a single, well-managed place has significantly improved the production readiness of ML in the industry.
- Most ML training and serving systems have some kind of a feature store even if it is not called that!
- Feature stores do not solve every problem
 - The most important characteristic of a feature store is its API
 - Different commercial and open source feature stores have different APIs

Feature Store(2)

A few basic features

- Store feature definitions
 - Usually these are stored as code that extracts the feature in a raw data format and outputs the feature data in the desired format
- Store feature values themselves
 - Ultimately, need to write features in a format that is easy to write, easy to read, and easy to use
 - will be largely determined by proposed use cases and most commonly is divided into ordered and unordered data
- Serve feature data
 - Provide access to feature data quickly and efficiently at a performance level suitable to the task
 - absolutely do not want expensive CPUs or accelerators stalled, waiting on the I/O of reading from feature store
- Coordinate metadata writes with the metadata system
 - To get the most out of feature store, should keep information about the data stored in it in the metadata system
 - helps model builders
 - Feature metadata is most useful for model developers
 - Pipeline metadata is most useful for ML reliability or production engineers

Feature Store(3)

Data Access Patterns requirements

- Consider asking the following questions as we think about what we need in a feature store:
 - Are we reading data in a particular order (log lines that are timestamped) or in no order that matters (a large collection of images in a cloud storage system)?
 - Will we read the features frequently or only when training a new model?
 - In particular, what is the ratio of bytes/records read to bytes/records written?
 - Is the feature data ingested once, never appended to, and seldom updated? Or is the data frequently appended to while older data is continuously deleted?
 - Can feature values be updated, or is the store append-only?
 - Are there special privacy or security requirements for the data we are storing?
 - In most cases, the extracted features of a dataset with privacy and use restrictions will also have privacy and use restrictions.
- After thinking about these questions, should be able to determine needs for a feature storage system
 - If lucky, will be able to use one of the existing commercial or open source feature stores on the market
 - If not, have to implement this functionality ourselves, whether we do it in an uncoordinated fashion or as part of a more coherent system

Feature Store(4)

Types

- Feature store falls into one of two buckets
 - Based on requirements of API and have a clearer understanding of data access needs
- Columns
 - The data is structured and decomposable into columns, not all of which will be used in all models
 - Typically, the data is also ordered in some way, often by time
 - column-oriented storage is the most flexible and efficient
- Blobs
 - binary large objects
 - the data is fundamentally unordered, mostly unstructured, and is best stored in a manner that's more efficient at storing a bunch of bytes
- Many feature stores will need to be replicated, partially or completely, in order to store data near the training and serving stacks
 - Since ML computation requirements are generally considerable for training, often prefer to replicate data to the place or places where can get the best value for training dollar
 - Having the feature store implemented as a service facilitates the management of this replication

Feature quality evaluation system

- As develop new features, need to evaluate what those features add to the quality of the overall model
 - in combination with existing features!
- General idea is to combine
 - the approaches of using slightly different models
 - A/B testing
 - and a model quality evaluation system
- in order to effectively evaluate the benefit of each new feature under development
 - can do this quickly and at relatively low cost.
- One common approach is to take an existing model and retrain it by using a single additional feature
 - then direct a fraction of our user requests to new model and evaluate its performance on a task
 - doing so can inform the choice of whether to keep the new feature or to eliminate it and try other ideas
- Even a feature that adds value may not be worth the cost to collect, process, store, and maintain it
 - For all but the most trivially obvious features, it is a good habit to calculate a rough return on investment (ROI) for every new feature added to the system
 - will help avoid useful features that are still more expensive than the value that they add



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Labeling

Pravin Y Pawar

Adapted from Designing Machine Learning Systems
by Chip Huyen

Labeling

- Despite the promise of unsupervised ML, most ML models in production today are supervised
 - means that they need labels to learn
- Performance of an ML model still depends heavily on the quality and quantity of the labeled data it's trained on
- There are tasks where data has natural labels or it's possible to collect natural labels on the fly
 - For predicting the click-through rate on an ad, labels are whether users click on an ad or not
 - Similarly, for recommendation systems, labels are whether users click on a recommended item or not
- However, for most tasks, natural labels are not available or not accessible
 - will need to obtain labels by other means
- The challenges of obtaining labels for data includes
 - label multiplicity problem
 - what to do when lack hand labeled data

Hand Labels

- Acquiring hand labels for data is difficult for many, many reasons!
- First, hand-labeling data can be **expensive**, especially if subject matter expertise is required
 - To classify whether a comment is spam, might be able to find 200 annotators on a crowdsourcing platform and train them in 15 minutes to label data
 - However, if want to label chest X-rays, need to find board-certified radiologists, whose time is limited and expensive
- Second, hand labeling poses a threat to **data privacy**
 - Hand labeling means that someone has to look at data, which isn't always possible if data has strict privacy requirements
 - For example, can't just ship patient's medical records or company's confidential financial information to a third-party service for labeling
 - In many cases, data might not even be allowed to leave organization, and might have to hire or contract annotators to label data on-premise
- Third, hand labeling is **slow**
 - For example, accurate transcription of speech utterance at phonetic level can take 400 times longer than the utterance duration
 - Slow labeling leads to slow iteration speed and makes model less adaptive to changing environments and requirements
 - If the task changes or data changes, have to wait for data to be relabeled before updating model

Label Multiplicity

- Often, to obtain enough labeled data, companies have to
 - rely on multiple annotators who have different levels of expertise
 - use data from multiple sources
- Leads to the problem of **label ambiguity or label multiplicity**:
 - what to do when there are multiple possible labels for a data instance
- Consider this simple task of entity recognition given to three annotators asking them to annotate all entities they can find
 - "Darth Sidious, known simply as the Emperor, was a Dark Lord of the Sith who reigned over the galaxy as Galactic Emperor of the First Galactic Empire."
 - Receive back three different solutions
 - Three annotators have identified different Entities
 - Which one should your model train on?

Annotator	# entities	Annotation
1	3	[Darth Sidious], known simply as the Emperor, was a [Dark Lord of the Sith] who reigned over the galaxy as [Galactic Emperor of the First Galactic Empire]
2	6	[Darth Sidious], known simply as the [Emperor], was a [Dark Lord] of the [Sith] who reigned over the galaxy as [Galactic Emperor] of the [First Galactic Empire].
3	4	[Darth Sidious], known simply as the [Emperor], was a [Dark Lord of the Sith] who reigned over the galaxy as [Galactic Emperor of the First Galactic Empire].

Label Multiplicity (2)

Rely on multiple annotators who have different levels of expertise

- Disagreements among annotators are extremely common!
 - The higher level of domain expertise required, the higher the potential for annotating disagreement
 - If one human-expert thinks the label should be A while another believes it should be B, how do we resolve this conflict to obtain one single ground truth?
 - If human experts can't agree on a label, what does human-level performance even mean?
- To minimize the disagreement among annotators, it's important to
- First, have a clear problem definition
 - For example, in the entity recognition task above, some disagreements could have been eliminated if clarified that in case of multiple possible entities, pick the entity that comprises the longest substring
 - means Galactic Emperor of the First Galactic Empire instead of Galactic Emperor and First Galactic Empire
- Second, need to incorporate that definition into training
 - to make sure that all annotators understand the rules

Label Multiplicity (3)

Use data from multiple sources

- Indiscriminately using data from multiple sources, generated with different annotators
 - without examining their quality can cause model to fail mysteriously
- Consider a case when trained a moderately good model with 100K data samples
 - ML engineers are confident that more data will improve the model performance
 - so spend a lot of money to hire annotators to label another million data samples
 - However, the model performance actually decreases after being trained on the new data
 - reason is that the new million samples were crowdsourced to annotators who labeled data with much less accuracy than the original data
 - can be especially difficult to remedy this if already mixed data and can't differentiate new data from old data
- Data Lineage
 - Good practice to keep track of the origin of each of data samples as well as its labels - data lineage
 - helps both in flagging potential biases in data as well as debug models

Handling the Lack of Hand Labels

- Because of the challenges in acquiring sufficient high-quality labels, many techniques have been developed to address the problems that result
- cover four of them: weak supervision, semi-supervision, transfer learning, and active learning

Method	How	Ground truths required?
Weak supervision	Leverages (often noisy) heuristics to generate labels	No, but a small number of labels are recommended to guide the development of heuristics
Semi-supervision	Leverages structural assumptions to generate labels	Yes. A small number of initial labels as seeds to generate more labels
Transfer learning	Leverages models pretrained on another task for your new task	No for zero-shot learning Yes for fine-tuning, though the number of ground truths required is often much smaller than what would be needed if you train the model from scratch.
Active learning	Labels data samples that are most useful to your model	Yes

Summaries for four techniques for handling the lack of hand labeled data.

Weak supervision

Approach

- If hand labeling is so problematic, what if we don't use hand labels altogether?
 - One approach that has gained popularity is **weak supervision**
 - One of the most popular open-source tools for weak supervision is **Snorkel**, developed at the Stanford
- The insight behind weak supervision is that people rely on heuristics, which can be developed with subject matter expertise, to label data.
 - For example, a doctor might use the following heuristics to decide whether a patient's case should be prioritized as emergent.
 - If the nurse's note mentions a serious condition like pneumonia, the patient's case should be given priority consideration
- Libraries like Snorkel are built around the concept of a **labeling function (LF)**: a function that encodes heuristics
- The above heuristics can be expressed by the following function

```
def labeling_function(note):
    if "pneumonia" in note:
        return "EMERGENT"
```
- After LFs are written, can apply them to the samples want to be labeled

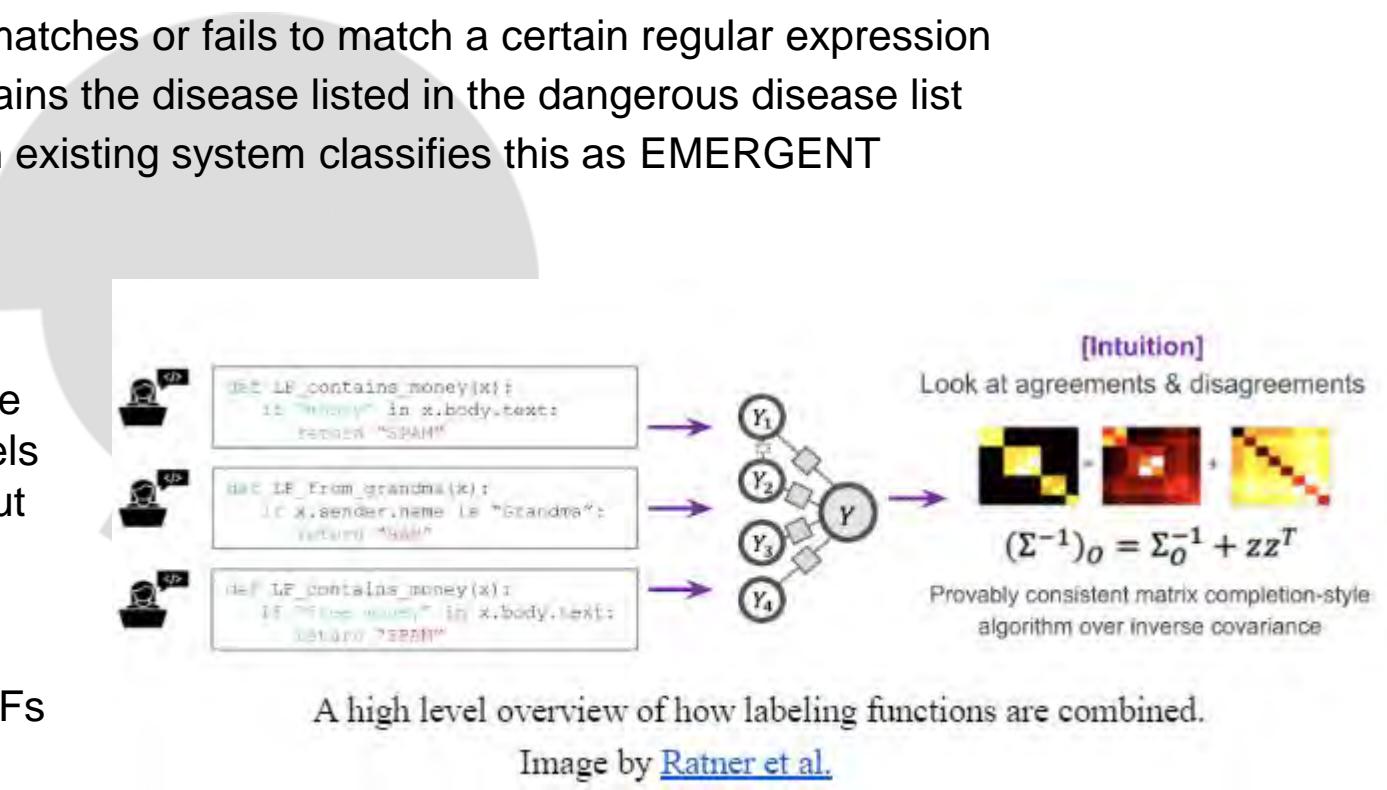
Weak supervision(2)

Heuristics and Labeling Functions

- LFs can encode many different types of heuristics
 - Keyword heuristic, such as the example above.
 - Regular expressions, such as if the note matches or fails to match a certain regular expression
 - Database lookup, such as if the note contains the disease listed in the dangerous disease list
 - The outputs of other models, such as if an existing system classifies this as EMERGENT

Because LFs encode heuristics, and heuristics are noisy, LFs are noisy!

- Multiple labeling functions might apply to the same data examples, and they might give conflicting labels
- One function might think a note is EMERGENT but another function might think it's not
- One heuristic might be much more accurate than another heuristic
- important to combine, denoise, and reweight all LFs to get a set of most likely-to-be-correct labels



Weak supervision(3)

Solution

- In theory, don't need any hand labels for weak supervision
 - However, to get a sense of how accurate LFs are, a small amount of hand labels is recommended
 - hand labels can help discover patterns in data to write better LFs
- Weak supervision can be especially useful when data has strict privacy requirements
 - only need to see a small, cleared subset of data to write LFs
 - which can be applied to the rest of data without anyone looking at it
- With LFs, subject matter expertise can be versioned, reused, and shared
 - Expertise owned by one team can be encoded and used by another team
 - If data changes or requirements change, can just reapply LFs to data samples
- Programmatic labeling
 - Approach of using labeling functions to generate labels for data

Weak supervision(4)

Hand labelling vs Programmatic labeling

Hand labeling	Programmatic labeling
Expensive: Especially when subject matter expertise required	Cost saving: Expertise can be versioned, shared, and reused across an organization
Non-private: Need to ship data to human annotators	Privacy: Create LFs using a cleared data subsample then apply LFs to other data without looking at individual samples.
Slow: Time required scales linearly with # labels needed	Fast: Easily scale from 1K to 1M samples
Non-adaptive: Every change requires re-labeling the data	Adaptive: When changes happen, just reapply LFs!

The advantages of programmatic labeling over hand labeling.

Weak supervision(5)

Drawbacks

- If heuristics work so well to label data, why do we need machine learning models?
 - One reason is that labeling functions might not cover all data samples
 - need to train ML models to generalize to samples that aren't covered by any labeling function
- Weak supervision is a simple but powerful paradigm. However, it's not perfect.
 - In some cases, the labels obtained by weak supervision might be too noisy to be useful
 - But it's often a good method to get started when want to explore the effectiveness of ML without wanting to invest too much in hand labeling upfront

Semi-supervision

- Semi-supervision leverages structural assumptions to generate new labels based on a small set of initial labels
 - Unlike weak supervision, semi-supervision requires an initial set of labels
- Self-training - A classic semi-supervision method
 - Start by training a model on existing set of labeled data, and use this model to make predictions for unlabeled samples
 - Assuming that predictions with high raw probability scores are correct, add the labels predicted with high probability to training set
 - then train a new model on this expanded training set
 - goes on until are happy with model performance
- Another semi-supervision method assumes that data samples that share similar characteristics share the same labels
 - The similarity might be obvious, such as in the task of classifying the topic of Twitter hashtags
 - In most cases, the similarity can only be discovered by more complex methods
 - use a clustering method or a K-nearest neighbor method to discover samples that belong to the same cluster

Semi-supervision(2)

- Perturbation-based method has gained popularity in recent years
 - based on the assumption that small perturbations to a sample shouldn't change its label
 - apply small perturbations to training samples to obtain new training samples
 - perturbations might be applied directly to the samples (e.g. adding white noise to images) or to their representations (e.g. adding small values to embeddings of words)
 - perturbed samples have the same labels as the unperturbed samples
- Semi-supervision is the most useful when the number of training labels is limited
 - One thing to consider when doing semi-supervision is how much of this limited amount should be used for evaluation
 - If evaluate multiple model candidates on the same test set and choose the one that performs best on the test set,
 - might have chosen a model that overfits the most on the test set
 - On the other hand, if choose models based on a validation set,
 - the value gained by having a validation set might be less than the value gained by adding the validation set to the limited training set

Transfer learning

Approach

- Refers to the family of methods where a model developed for a task is reused as the starting point for a model on a second task
- First, the base model is trained for a base task such as language modeling
 - The base task is usually a task that has cheap and abundant training data
 - Language modeling is a great candidate because it doesn't require labeled data
 - Can collect any body of text — books, Wikipedia articles, chat histories — and the task is: given a sequence of tokens, predict the next token
- Then use this pretrained base model on the task that you're interested in, - called a downstream task
 - such as sentiment analysis, intent detection, question answering, etc
 - In zero-shot learning scenarios, might be able to use the base model on a downstream task directly
 - In many cases, might need to fine-tune the base model
 - Fine-tuning means making small changes to the base model, which can be continuing training the entire base model or a subset of the base model on data from a given downstream task

Transfer learning(2)

Discussion

- Especially appealing for tasks that don't have a lot of labeled data
 - Even for tasks that have a lot of labeled data, using a pretrained model as the starting point can
 - often boost the performance significantly compared to training from scratch
- Transfer learning has gained a lot of interest in recent years
 - enabled many applications that were previously impossible due to the lack of training samples
 - A non-trivial portion of ML models in production today are the results of transfer learning,
 - including object detection models that leverage models pretrained on ImageNet and text classification models that leverage pretrained language models such as BERT or GPT-3.
 - lowers the entry barriers into ML, as it helps reduce the upfront cost needed for labeling data to build ML applications
- A trend that has emerged in the last five years is that usually,
 - the larger the pretrained base model, the better its performance on downstream tasks
- Large models are expensive to train
 - Based on the configuration of GPT-3, it's estimated that the cost of training this model is in the tens of millions USD
 - Many have hypothesized that in the future, only a handful of companies can afford to train large pretrained models
 - The rest of the industry will use these pretrained models directly or fine-tune them for their specific needs

Active learning

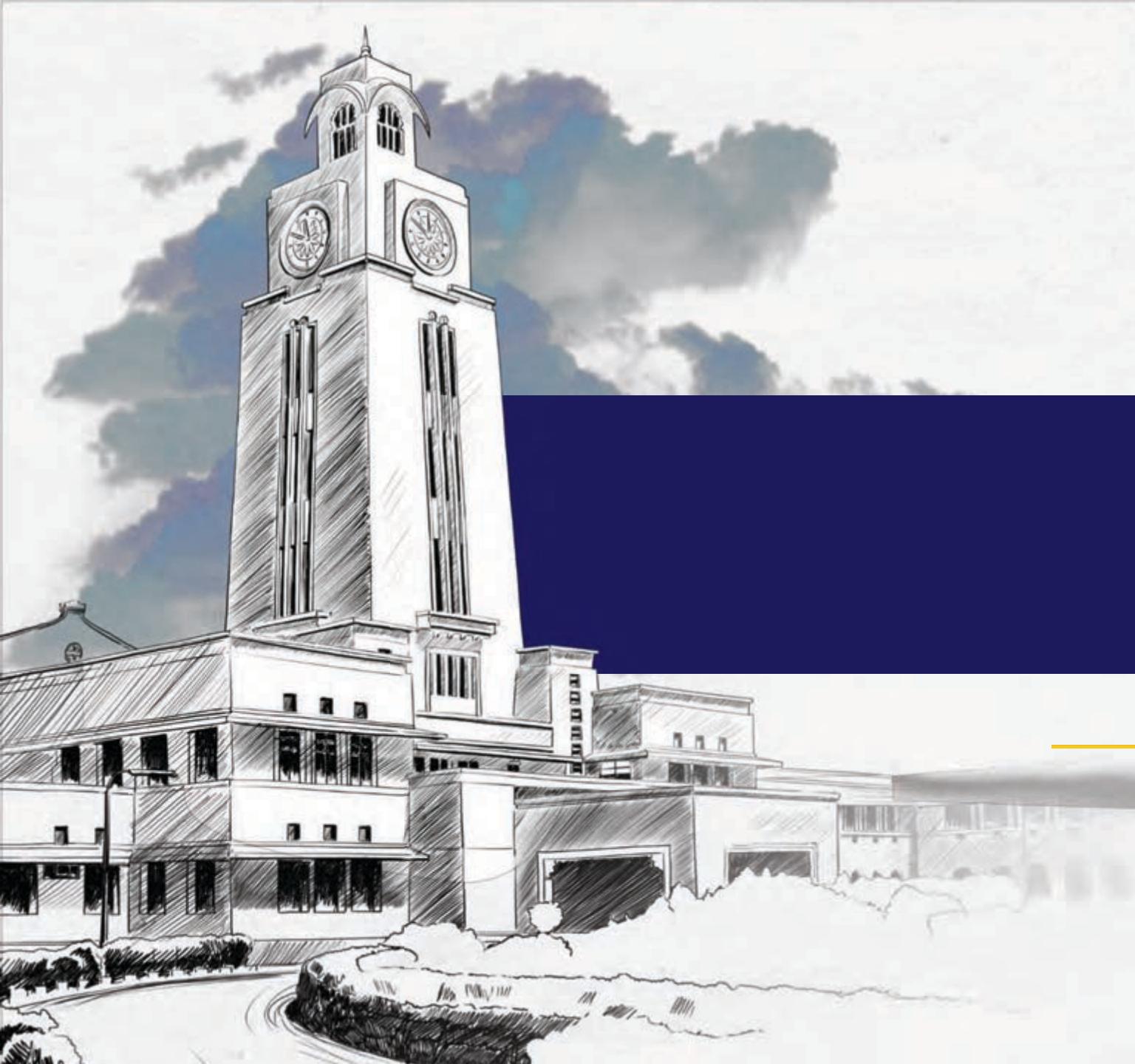
Approach

- Method for improving the efficiency of data labels
 - The hope here is that ML models can achieve greater accuracy with fewer training labels if they can choose which data samples to learn from
- Aka query learning
 - because a model (active learner) sends back queries in the form of unlabeled samples to be labeled by annotators (usually humans)
- Instead of randomly labeling data samples, label the samples that are most helpful to models according to some heuristics
 - The most straightforward heuristic is uncertainty measurement — label the examples that model is the least certain about hoping that they will help model learn the decision boundary better
 - For example, in the case of classification problems where model outputs raw probabilities for different classes, it might choose the data examples with the lowest probabilities for the predicted class
 - Another common heuristic is based on disagreement among multiple candidate models - query-by-committee
 - need a committee of several candidate models, which are usually the same model trained with different sets of hyperparameters
 - each model can make one vote for which examples to label next, which it might vote based on how uncertain it is about the prediction
 - then label the examples that the committee disagrees on the most



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Labels Systems

Pravin Y Pawar

Adapted from Reliable Machine Learning
by Cathy Chen

Human-Generated Labels

- Lets Turn attention to the large classes of problems requiring human annotations to provide the model training data
 - For example, building a system to analyze and interpret human voice
 - needs human annotation to ensure that the transcription is accurate and to understand what the speaker meant
 - Image analyses and understanding
 - often needs example annotated images for image classifications or detection problems
- Getting these human annotations at the scale needed to train an ML model is challenging
 - from both implementation and cost perspectives
 - Effort must be dedicated to designing efficient systems to produce these annotations
- Humans need to be trained on the task
 - Because of the large cost associated with acquiring human-generated labels,
 - training systems should be designed to get as much mileage from them as possible
- While some kinds of tremendously complex data can best be labeled by humans,
 - other types of data definitely cannot be labeled by humans at all
 - abstract data with high dimensionality that makes it difficult for a human to determine the correct answer quickly
 - Sometimes humans can be provided with augmentation software
 - to assist in these tasks, but other times they are just the wrong option for performing the labeling

Annotation Workforces

- The first question that often comes up with human annotation problems is **who will do the labeling**
 - **question of scale and equity!**
- For simpler models, for which a small amount of data is sufficient to train the model,
 - typically the engineer building the model will do their own labeling, often with hacky, homebuilt tools
- For more complex models, dedicated annotation teams are used to provide human annotations at a scale not otherwise possible
 - dedicated teams can be collocated with the model builder or remotely provided by third-party annotation providers
 - teams range in size from a single person to hundreds of people, all generating annotated data for a single model
- The Amazon Mechanical Turk service was the original platform used for this
 - but since then a proliferation of crowdsourcing platforms and services have developed
 - Some of these services use paid volunteers, and others use teams of employees to label the data
- Cost, quality, and consistency trade-offs arise in the choice of labeling
 - Crowdsourced labeling often requires additional effort to verify quality and consistency, but paid labeling staff can be expensive
 - The costs for these annotation teams can easily exceed the computational costs of training the models

Measuring Human Annotation Quality

- As the quality of any model is only as good as the data used to train the model,
 - quality must be designed into the system from the start
 - becomes increasingly true as the size of the annotation team grows
- Quality can be achieved in multiple ways depending on the task, but the most frequent techniques used include the following:
 - Multiple labeling (also called consensus labeling)
 - The same data is given to multiple labelers to check for agreement among them.
 - Golden set test questions
 - Trusted labelers (or the model builder) produce a set of test questions that are randomly included in the unlabeled data to evaluate the quality of the produced labels
 - A separate QA step
 - A fraction of the labeler data is reviewed by a more trusted QA team

Measuring Human Annotation Quality (2)

- Once measured, quality metrics can be improved!
- Quality issues are best addressed by managing the annotation team with humility
- and by understanding that they will produce higher-quality results when they have the following:
 - More training and documentation
 - Recognition for quality and not just throughput
 - A variety of tasks over the workday
 - Easy-to-use tools
 - Tasks with a balanced set of answers
 - An opportunity to provide feedback on the tools and instructions
- However, even the best, most conscientious labeler will miss things occasionally
 - so processes should be designed to detect or accept these occasional errors

An Annotation Platform

- A labeling platform organizes the flow of data to be annotated and the results of the annotations
 - while providing quality and throughput metrics of the overall process
- Are primarily work-queuing systems to divide the annotation work among the annotators
 - The actual labeling tool that allows the labelers to view the data and provide their annotations
- With a team or organization that is working on multiple models simultaneously,
 - the same annotation team may be shared among multiple annotation projects
 - each annotator might have different sets of skills (e.g., language skills)
 - queuing systems can be relatively complex and require careful design to avoid problems such as scalability issues or queue starvation
- Pipelines enabling the output of one annotation task to serve as the input to another can be useful for complex workflows
 - Quality measurement should be designed into the system from the start
 - so project owners can understand labeling throughput and quality of all their annotation tasks

An Annotation Platform(2)

- Although historically many companies have implemented their own labeling platforms with their own set of these features,
 - many options exist for prebuilt labeling platforms
- The major cloud providers and many smaller startups offer labeling platform services
 - that can be used with arbitrary annotation workforces
- many annotation workforce providers have their own platform options that can be used
- Rapidly changing area with new features being added to existing platforms all the time
 - Publicly available tools are moving beyond simple queuing systems
 - are starting to provide dedicated tools for common tasks, including advanced features like AI-assisted labeling
- In many cases the most sensible place to store completed labels is in the feature store

Active Learning and AI-Assisted Labeling

- Active learning techniques
 - focus the annotation effort on the cases in which the model and the human annotators disagree or the model is most uncertain
 - thereby improve overall label quality
- A semi-supervised system
 - allows the modeler to bootstrap the system with weak heuristic functions that imperfectly predict the labels of some data
 - then use humans to train a model that takes these imperfect heuristics to high-quality training data
 - can be particularly valuable for problems with complex, frequently changing category definitions requiring models to be retrained quickly and frequently
- Efficient annotation techniques for particularly complex labeling tasks is an ongoing field of research
 - a quick review of available tools from cloud and annotation providers is well worth time
 - are often adding new capabilities for AI-assisted annotations

Documentation and labeler training systems

- Most commonly overlooked parts of an annotation platform!
- While labeling instructions often start simply
 - inevitably get more complex as data is labeled and various corner cases are discovered
- Labeling definitions and directions should be updated as new corner cases are discovered
 - annotation and modeling teams should be notified about the changes
 - If the changes are significant, previously labeled data might have to be re-annotated to correct data labeled with old instructions
- Annotation teams often have significant turnover
 - investing in training for using annotation tools and for understanding labeling instructions will almost always give big wins in label quality and throughput



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Data Augmentation

Pravin Y Pawar

Adapted from Designing Machine Learning Systems
by Chip Huyen

Data Augmentation

- A family of techniques that are used to increase the amount of training data
 - Traditionally, used for tasks that have limited training data, such as in medical imaging projects
 - However, in the last few years, shown to be useful even when have a lot of data
 - because augmented data can make models more robust to noise and even adversarial attacks
- Data augmentation has
 - become a standard step in many computer vision tasks
 - finding its way into natural language processing (NLP) tasks
- Three main types of data augmentation:
 - simple label-preserving transformations,
 - perturbation, which is a term for “adding noises”,
 - data synthesis

Simple Label-Preserving Transformations

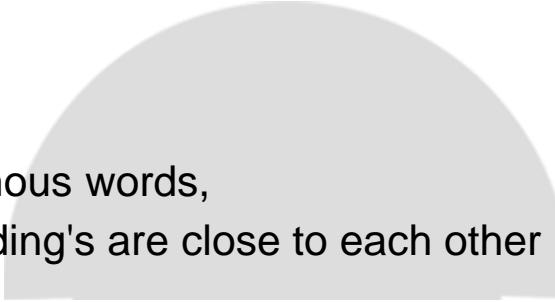
Computer vision

- In computer vision, the simplest data augmentation technique is to randomly modify an image while preserving its label
 - can modify the image by
 - cropping,
 - flipping,
 - rotating,
 - inverting (horizontally or vertically),
 - erasing part of the image, and more
 - makes sense because a rotated image of a dog is still a dog
- Common ML frameworks like PyTorch and Keras both have support for image augmentation
- According to Krizhevsky et al., in their legendary AlexNet paper,
 - “the transformed images are generated in Python code on the CPU while the GPU is training on the previous batch of images. So these data augmentation schemes are, in effect, computationally free.”

Simple Label-Preserving Transformations

NLP

- In NLP, can randomly replace a word with a similar word,
 - assuming that this replacement wouldn't change the meaning or the sentiment of the sentence
- Similar words can be found
 - either with a dictionary of synonymous words,
 - or by finding words whose embedding's are close to each other in a word embedding space



Original sentences	I'm so happy to see you.
Generated sentences	I'm so glad to see you. I'm so happy to see y'all . I'm very happy to see you.

Three sentences generated from an original sentence by replacing a word with another word with similar meaning.

- This type of data augmentation is a quick way to double, even triple training data

Perturbation

- Perturbation is also a label-preserving operation,
 - but sometimes, used to trick models into making wrong predictions
- Neural networks, in general, are sensitive to noise.
 - In the case of computer vision, this means that by adding a small amount of noise to an image can cause a neural network to misclassify it.
- Adversarial attacks
 - Using deceptive data to trick a neural network into making wrong predictions
 - Adding noise to samples to create adversarial samples is a common technique
 - Success of adversarial attacks is especially exaggerated as the resolution of images increases

Perturbation

- Adversarial augmentation
 - Adding noisy samples to training data can help models recognize the weak spots in their learned decision boundary and improve their performance
 - Noisy samples can be created by either adding random noise or by a search strategy
 - Moosavi-Dezfooli et al. proposed an algorithm, called DeepFool, that finds the minimum possible noise injection needed to cause a misclassification with high confidence
- Adversarial augmentation is less common in NLP but perturbation has been used to make models more robust
 - One of the most notable examples is BERT

Data Synthesis

- Since collecting data is expensive and slow with many potential privacy concerns
 - it'd be a dream if could sidestep it altogether and train models with synthesized data
- Still far from being able to synthesize all training data,
 - it's possible to synthesize some training data to boost a model's performance
- In NLP,
 - templates can be a cheap way to bootstrap model.

A template might look like:

“Find me a [CUISINE] restaurant within [NUMBER] miles of [LOCATION].” With lists of all possible cuisines, reasonable numbers (you would probably never want to search for restaurants beyond 1000 miles), and locations (home, office, landmarks, exact addresses) for each city, you can generate thousands of training queries from a template.

Template	Find me a [CUISINE] restaurant within [NUMBER] miles of [LOCATION].
Generated queries	<ul style="list-style-type: none">• Find me a Vietnamese restaurant within 2 miles of my office.• Find me a Thai restaurant within 5 miles of my home.• Find me a Mexican restaurant within 3 miles of Google headquarters.

Three sentences generated from a template

Data Synthesis

Computer vision

- In computer vision, a straightforward way to synthesize new data
 - is to combine existing examples with discrete labels to generate continuous labels
- Mixup
 - Consider a task of classifying images with two possible labels: DOG (encoded as 0) and CAT (encoded as 1)
 - From example x_1 of label DOG and example x_2 of label CAT, can generate x' such as:
$$x' = \gamma x_1 + (1 - \gamma)x_2$$
 - The label of x' is a combination of the labels of x_1 and x_2 :
$$\gamma * 0 + (1 - \gamma) * 1$$
- Research showed mixup
 - improves models' generalization
 - reduces their memorization of corrupt labels
 - increases their robustness to adversarial examples
 - and stabilizes the training of generative adversarial networks
- Using neural networks to synthesize training data is an exciting approach
 - actively being researched but not yet popular in production



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Engineering Good Features

Pravin Y Pawar

Adapted from Designing Machine Learning Systems
By Chip Huyen

Engineering Good Features

- Generally adding more features leads to better model performance
 - List of features used for model in production only grows over time
 - However, more feature doesn't always mean better model performance
- Two factors might need to consider when evaluating whether a feature is good for model
 - Feature Importance to the model
 - Feature Generalization to unseen data

Too many features

Can be bad both during training and serving model because

- More features means more opportunities for data leakage
- Too many features can cause overfitting
- Too many features can increase memory required to serve a model,
 - which in turn, might require to use a more expensive machine/instance to serve model
- Too many features can increase inference latency when doing online prediction
 - Especially if needed to extract these features from raw data for predictions online
- Useless features become technical debts
 - Whenever data pipeline changes, all the affected features need to be adjusted accordingly
- Feature removal
 - Might help models learn faster if not useful features are removed, prioritizing good features
 - Can store removed features to add them back later
 - Can store general feature definitions to reuse and share across teams in organization
 - Feature stores can be used for the same!

Feature Importance

Many methods to measure importance

- Use built-in feature importance functions
 - For classical ML algorithms like boosted gradient trees implemented by XGBoost
- For model agnostic methods, look at
 - SHAP (SHapely Additive explanations)
 - InterpretML
- Exact algorithms for feature importance measurement is complex
 - A features importance to a model is measured by how much that model's performance deteriorates if that feature is removed from model
 - SHAP not only measures a importance but also measures each features contribution to a models specific prediction
- Often a small number of features accounts for large portion of models feature importance
- Facebook team while measuring importance of click-through rate prediction model, found out that
 - top 10 features are responsible for about half of models total feature importance
 - Last 300 features contributes less than 1% of feature importance

Feature Generalization

- Goal of ML model is to make correct predictions on unseen data,
 - Features used for model should generalize to unseen data
 - Not all features generalize equally!
 - To predict whether a comment is spam, identifier of each comment is not generalizable but username of user who posts comment might be more meaningful
- Measuring feature generalization is lot less scientific than measuring feature importance
 - Requires both intuition and subject matter expertise on top of statistical knowledge
- Two aspects might want to consider with regards to generalization
 - Feature coverage
 - Distribution of feature values

Feature coverage

- Coverage is percentage of samples that has values for feature in data
 - Fewer values that are missing, higher the coverage
 - Feature that appears in a very small percentage of data, not very generalizable
 - To predict whether a person will purchase house in next 12 months, number of children values is present only for 1% of data, feature might not be useful
- Some features can still be useful even if they are missing in most of data
 - Common when missing values are present not at random
- Coverage differs wildly between different slices of data or in even in same slice of date over time
 - If coverage differs a lot between train and test – indication of different distribution of data
 - Need to investigate whether the way used to split data makes sense and whether this feature is cause of data leakage

Distribution of feature values

- Feature might hurt models performance
 - If set of values appears in seen data (train split) has no overlap with set of values appears in unseen data (test split / real time data)
- For example,
- To build a model to estimate time it will take for given taxi ride
 - Might retrain model every week and use data from last six days to predict ETA for today
 - DAY_OF_WEEK might be useful feature as traffic on weekdays is worse than weekends
 - Feature coverage is 100% - as its present for all data records
 - In train split, values are from MONDAY to SATURDAY
 - In test split, value is SUNDAY
 - If used as is it won't generalize the test split and hurt models performance
 - HOUR_OF_DAY can be a good feature
 - Time in day affects traffic and range of values in train split overlaps with test split 100%

Specificity

- When considering features generalization, trade-off between generalization and specificity comes in
- For example,
- Might realize that traffic during an hour only changes depending on whether hour is rush hour
 - So generate IS_RUSH_HOUR feature to have value as 1 if hour is between 7 to 9 am or between 4 to 6 pm
 - IS_RUSH_HOUR is more generalizable but less specific than HOUR_OF_DAY
 - Using IS_RUSH_HOUR without HOUR_OF_DAY might cause models to lose important information about hour



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

ML Experiment Tracking

Pravin Y Pawar

Adapted from
[ML Experiment Tracking: What It Is, Why It Matters, and How to Implement It](#)
by Jakub Czakon

a story that have heard too many times

"... So far we have been doing everything manually and sort of ad hoc.

Some people are using it, some people are using that, it's all over the place.

We don't have anything standardized.

But we run many projects, the team is growing, and we are scaling pretty fast.

So we run into a lot of problems. How was the model trained? On what data? What parameters we used for different versions? How can we reproduce them?

We just feel the need to control our experiments..."

– unfortunate Data Scientist.

A lot of experiments

- And the truth is, when develop ML models, will run **a lot of experiments**
- Those experiments may:
 - use different models and model hyper parameters
 - use different training or evaluation data,
 - run different code (including this small change that you wanted to test quickly)
 - run the same code in a different environment (not knowing which PyTorch or Tensorflow version was installed)
 - And as a result, they can produce completely different evaluation metrics.
- Keeping track of all that information can very quickly become really hard
- Especially
 - if want to organize and compare those experiments
 - feel confident that you know which setup produced the best result
- This is where **ML experiment tracking** comes in!

ML experiment tracking

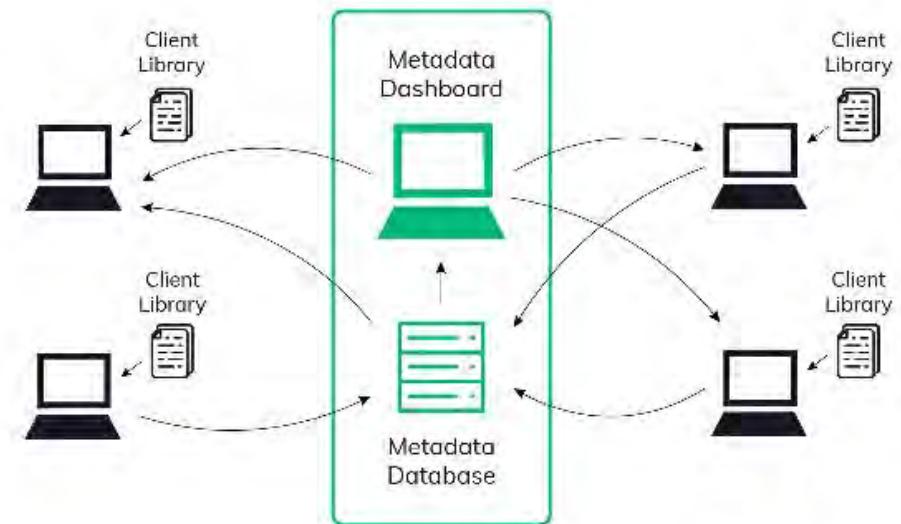
What?

- Experiment tracking is the process of saving all experiment related information that you care about for every experiment you run
- This “metadata you care about” will strongly depend on your project, but it may include:
 - Scripts used for running the experiment
 - Environment configuration files
 - Versions of the data used for training and evaluation
 - Parameter configurations
 - Evaluation metrics
 - Model weights
 - Performance visualizations (confusion matrix, ROC curve)
 - Example predictions on the validation set (common in computer vision)
- Of course, you want to have this information available after the experiment has finished,
 - but ideally, you’d like to see some of it as your experiment is running as well.

ML experiment tracking

Components

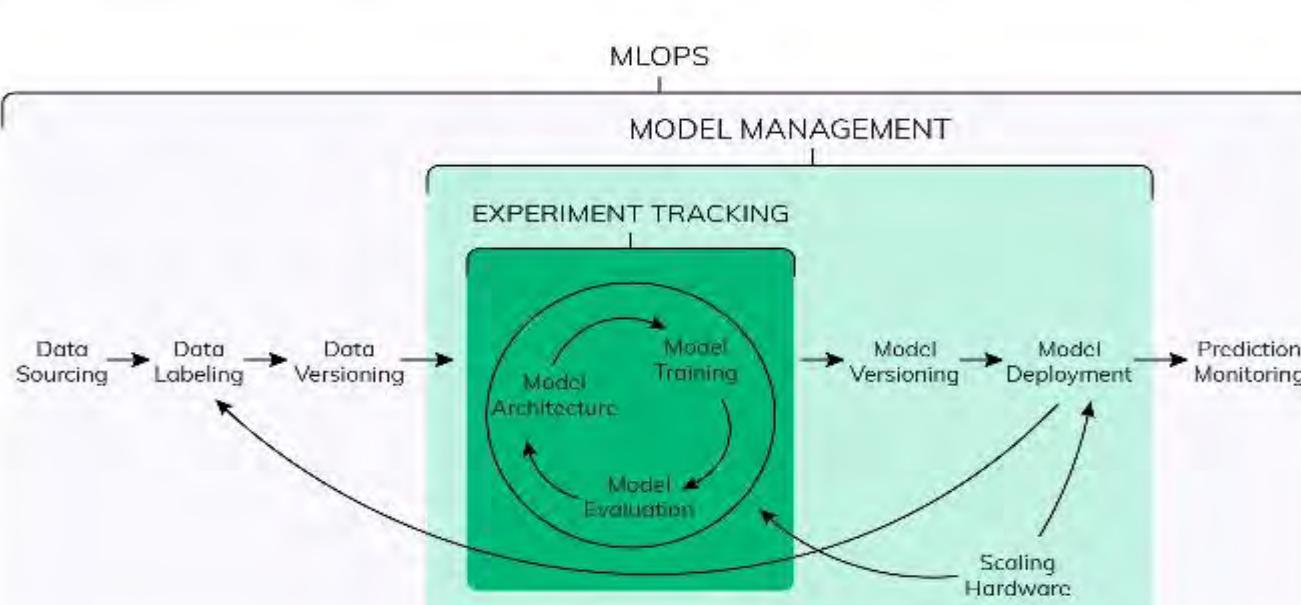
- To do experiment tracking properly, need some sort of a system that deals with all this metadata
- Typically, such a system will have 3 components:
 - Experiment database: A place where experiment metadata is stored and can be logged and queried
 - Experiment dashboard: A visual interface to your experiment database - A place where can see experiment metadata
 - Client library: Which gives methods for logging and querying data from the experiment database
- Of course, you can implement each component in many different ways, but the general picture will be very similar



Experiment tracking vs ML model management vs MLOps

isn't experiment tracking like MLOps or something?

- Experiment tracking (also referred to as experiment management) is a part of MLOps:
 - a larger ecosystem of tools and methodologies that deals with the operationalization of machine learning
- MLOps deals with every part of ML project lifecycle
 - from developing models by scheduling distributed training jobs, managing model serving,
 - monitoring the quality of models in production, and re-training those models when needed



Experiment tracking vs ML model management vs MLOps(2)

how is experiment tracking different from ML model management?

- Experiment tracking focuses on the iterative model development phase
 - when you try many things to get your model performance to the level you need
- **ML model management starts when models go to production:**
 - streamlines moving models from experimentation to production
 - helps with model versioning
 - organizes model artifacts in an ML model registry
 - helps with testing various model versions in the production environment
 - enables rolling back to an old model version if the new one seems to be going crazy
 - But not every model gets deployed.
- **Experiment tracking is useful even if your models don't make it to production (yet).**
 - And in many projects, especially those that are research-focused, they may never actually get there
 - But having all the metadata about every experiment you run ensures that you will be ready when this magical moment happens.

Why does experiment tracking matter?

4 ways in which experiment tracking can actually improve ML workflow

neptune.ai

4 ways **experiment tracking** can make your **workflow better**

- 01.**
One source of truth
All your experiments and models are organized in a single place.
- 02.**
Automated process
It lets you compare experiments, analyze results, and debug model training with little extra work.
- 03.**
Better collaboration
You can see what everyone is doing, share results, and access experiment data programmatically.
- 04.**
Live monitoring
You can see your ML runs live and manage experiments from anywhere and anytime.

neptune.ai/blog/ml-experiment-tracking

All ML experiments are organized in a single place

- There are many ways to run ML experiments or model training jobs:
 - Private laptop
 - PC at work
 - A dedicated instance in the Cloud
 - University cluster
 - Kaggle kernel or Google Colab
 - And many more.
- Sometimes just want to test something quickly and run an experiment in a notebook
- Sometimes want to spin up a distributed hyper parameter tuning job
- Either way, during the course of a project (especially when there are more people working on it),
 - can end up having your experiment results scattered across many machines

ML experiments are organized in a single place(2)

- With the experiment tracking system, all of experiment results are logged to one experiment repository by design
 - keeping all of experiment metadata in a single place, regardless of where its run, makes experimentation process so much easier to manage
- Specifically, a centralized experiment repository makes it easy to:
 - Search and filter experiments to find the information you need quickly
 - Compare their metrics and parameters with no additional work
 - Drill down and see what exactly it was that you tried (code, data versions, architectures)
 - Reproduce or re-run experiments when you need to
 - Access experiment metadata even if you don't have access to the server where you ran them
- Additionally, **can sleep peacefully** knowing that all the ideas tried are safely stored, and can always go back to them later

Compare experiments, analyze results, debug model training with little extra work

- Whether debugging training runs, looking for improvement ideas, or auditing current best models, comparing experiments is important
- But when don't have any experiment tracking system in place:
 - the way you log things can change,
 - you may forget to log something important
 - you may simply lose some information accidentally
- In those situations, something as simple as comparing and analyzing experiments can get difficult or even impossible
- With an experiment tracking system, experiments are stored in a single place,
 - follow the same protocol for logging them, so those comparisons can go really deep

Benefits of tracking

- Proper experiment tracking makes it easy to:
 - Compare parameters and metrics
 - Overlay learning curves
 - Group and compare experiments based on data versions or parameter values
 - Compare Confusion Matrices, ROC curves, or other performance charts
 - Compare best/worst predictions on test or validation sets
 - View code diffs (and/or notebook diffs)
 - Look at hardware consumption during training runs for various models
 - Look at prediction explanations like Feature Importance, SHAP or Lime
 - Compare rich-format artifacts like video or audio
 - ... Compare anything else you logged
- Modern experiment tracking tools will give many of those comparison features (almost) for free
 - Some tools even go as far as to automatically find diffs between experiments or show which parameters have the biggest impact on model performance
- When have all the pieces in one place, might be able to find new insights and ideas
 - just by looking at all the metadata logged
 - especially true when you are not working alone!

What should be tracked in any ML experiment?

- Things that should be keep track of regardless of the project are:
 - Code:
 - preprocessing, training and evaluation scripts, notebooks used for designing features, other utilities.
 - All the code that is needed to run (and re-run) the experiment
 - Environment:
 - The easiest way to keep track of the environment is to save the environment configuration files like `Dockerfile` (Docker), `requirements.txt` (pip) or `conda.yml` (conda)
 - Can also save the Docker image on Docker Hub, but I find saving configuration files easier
 - Data:
 - saving data versions (as a hash or locations to data files) makes it easy to see what your model was trained on
 - can also use modern data versioning tools like DVC (and save the .dvc files to your experiment tracking tool)
 - Parameters:
 - saving your run configuration is absolutely crucial
 - Metrics:
 - logging evaluation metrics on train, validation, and test sets for every run is pretty obvious
- Keeping track of those things will let you reproduce experiments, do basic debugging, and understand what happened at a high-level.
 - That said, can always log more things to gain even more insights.

How to set up experiment tracking?

- There are (at least) a few options. The most popular being:
 - Spreadsheets + naming conventions
 - Versioning configuration files with Github
 - Using modern experiment tracking tools

Spreadsheets + naming conventions

(but please don't use it)

- A common approach
 - Simply create a big spreadsheet to put all of the information (metrics, parameters, etc) and a directory structure
 - where things are named in a certain way
 - names usually end up being really long like 'model_v1_lr01_batchsize64_no_preprocessing_result_accuracy082.h5'
 - Whenever an experiment is run, look at the results and copy them to the spreadsheet
- In some situations, it can be just enough to solve experiment tracking problems
 - may not be the best solution but it is quick and simple
 - ...things can fall apart really quickly!

Spreadsheets + naming conventions(2)

(but please don't use it)

- A few major reasons why tracking experiments in spreadsheets doesn't work for many people:
 - have to remember to track them
 - If something doesn't happen automatically, things get messy, especially with more people involved
 - have to be sure that you or your team will not overwrite things in the spreadsheet by accident
 - Spreadsheets are not easy to version, so if this happens, you are in trouble
 - have to remember to use the naming conventions
 - If someone on your team messes this up, you may not know where the experiment artifacts (model weights, performance charts) for the experiments you ran are.
 - have to back up your artifact directories (remember that things break)
 - When your spreadsheet grows, it becomes less and less usable
 - Searching for things and comparing hundreds of experiments in a spreadsheet (especially if you have multiple people that want to use it at the same time) is not a great experience.

Versioning configuration files with Github

Can version metadata files in GitHub

- Another option is to version all of experiment metadata in Github
 - to commit metrics, parameters, charts, and whatever want to keep track of to Github when running experiment
 - can be done with post-commit hooks where you create or update some files (configs, charts, etc) automatically after experiment finishes
- ... Github wasn't built for ... machine learning
- It can work in some setups but:
 - .git and Github wasn't built for comparing machine learning objects
 - Comparing more than two experiments is not going to work. Compare in .git systems was designed for comparing two branches, master and develop, for example. If you want to compare multiple experiments, take a look at metrics and overlay learning curves you are out of luck.
 - Organizing many experiments is difficult (if not impossible). You can have branches with ideas or a branch per experiment but the more experiments you run the less usable it becomes
 - You will not be able to monitor your experiments live, the information will be saved after your experiment is finished.

Using modern experiment tracking tools

- While can try and adjust general tools to work for machine learning experiments,
 - could just use one of the solutions built specifically for tracking, organizing, and comparing experiments
- Experiment is logged to a central experiment database and displayed in the experiment dashboard,
 - where it can be searched, compared, and drill down to whatever information needed
- Today there are at least a few good tools for experiment tracking!
- Designed to treat machine learning experiments as first-class citizens, and they will always:
 - be easier to use for a machine learning person than general tools
 - have better integrations with the ML ecosystem
 - have more experiment-focused features than the general solutions



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

ML Model Registry – what?

Pravin Y Pawar

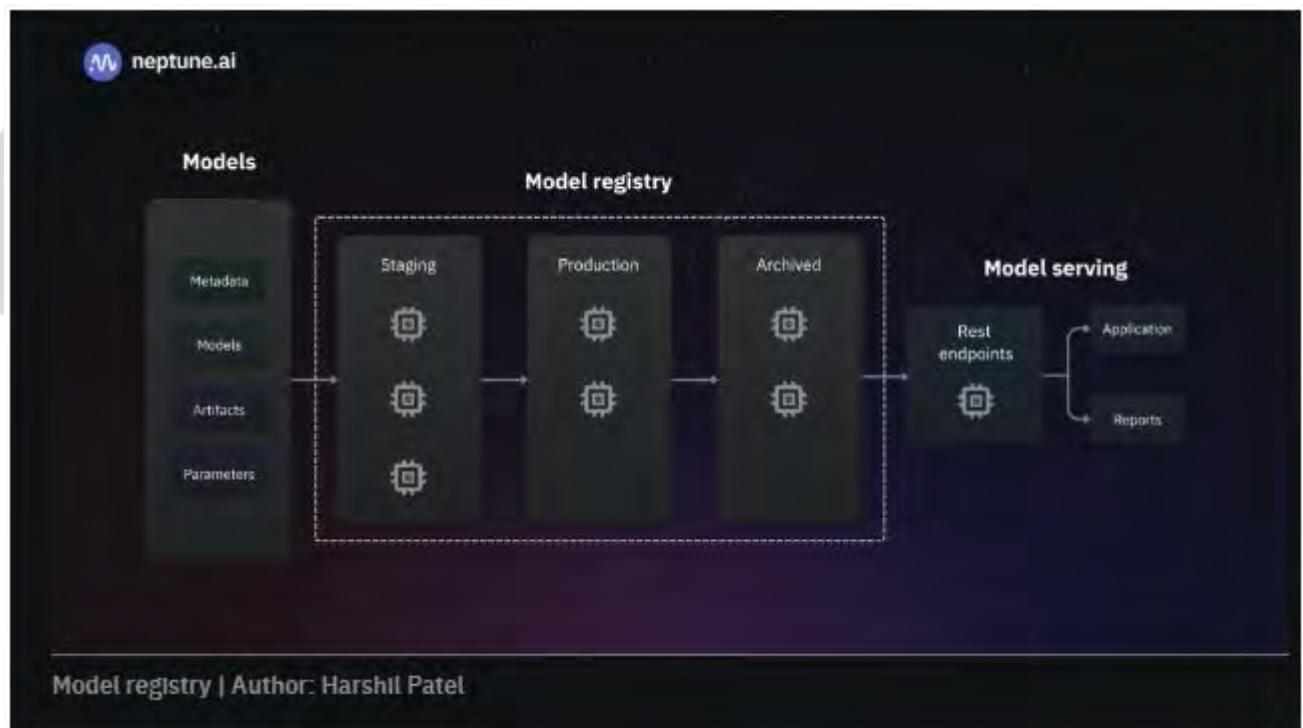
Adapted [from ML Model Registry](#)
By Stephen Oladele

Questions from cross-functional team

- Where can we find the best version of this model so we can audit, test, deploy, or reuse it?
- How was this model trained?
- How can we track the docs for each model to make sure they are compliant and people can know the necessary details about it including the metadata?
- How can we review models before they are put to use or even after they have been deployed?
- How can we integrate with tools and services that make shipping new projects easier?
- They want to understand
 - what is running in production,
 - how to improve it
 - or roll back to previous versions
- It makes perfect sense!

What is a model registry?

- An ML model registry serves as a centralized repository, enabling effective model management and documentation
- Allows for
 - clear naming conventions,
 - comprehensive metadata,
 - and improved collaboration between data scientists and operations teams,
 - ensuring smooth deployment and utilization of trained models
- A data scientist can push trained models to the model registry
 - Once in the registry, models are ready to be tested, validated, and deployed to production



Model registry vs model repository

- Model Repository is a storage location for machine learning models
- Model Registry is a more comprehensive system that tracks and manages the full lifecycle of machine learning models.
- However, both are often used interchangeably, and the specific definitions may vary depending on the context or the tools and platforms being used.

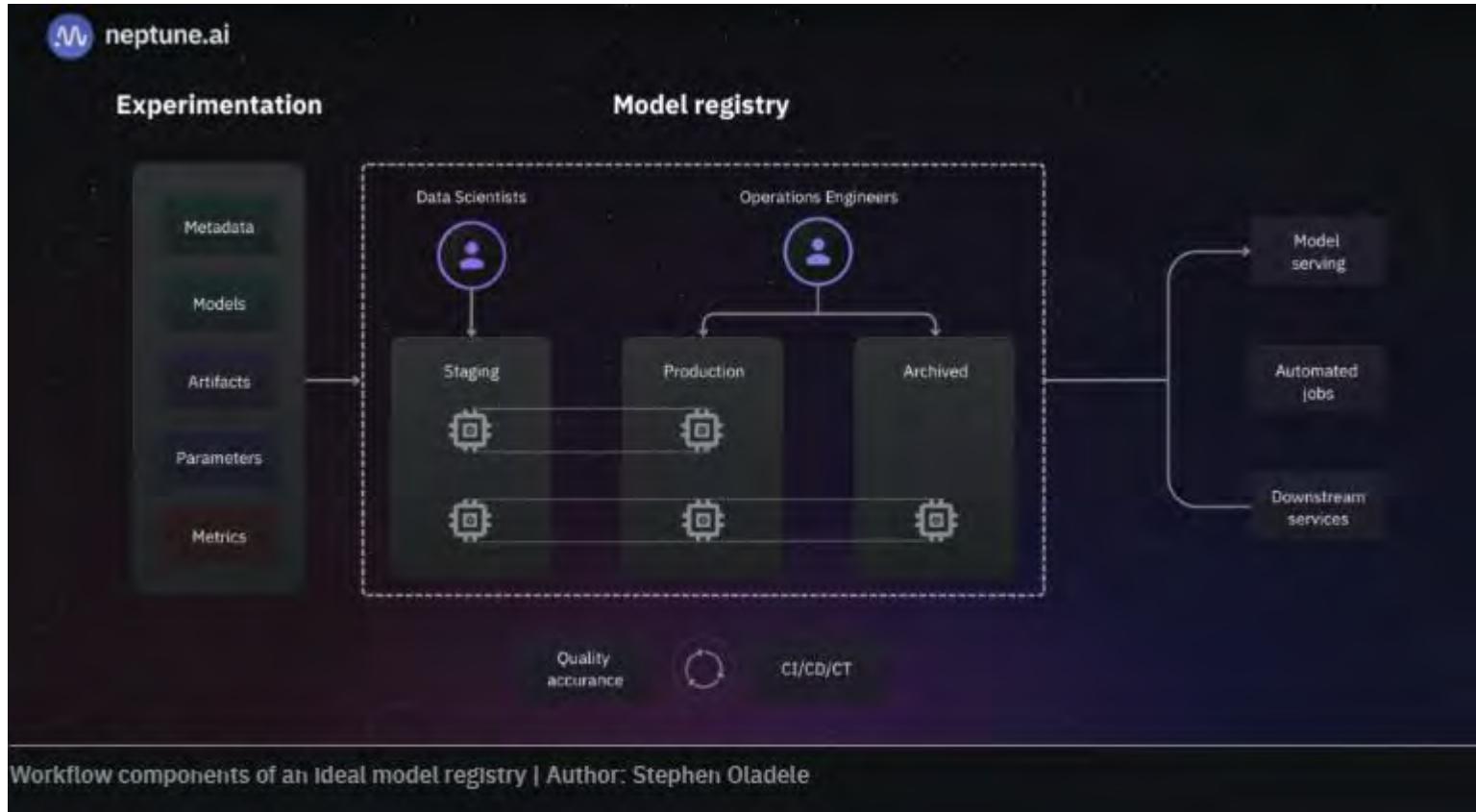
Feature	Model Repository	Model Registry
Storage	Stores machine learning models in a file format	Stores machine learning models and associated metadata
Version control	No	Yes
Lineage tracking	No	Yes
Access control	No	Yes
Other features	None	Can track the performance of models over time, ensure reliability and reproducibility of models, and protect sensitive models from unauthorized access

Model registry vs Experiment tracking

- Model registry has to integrate with the Experiment management system (which tracks the experiments) to register models from various experiment runs to make them easier to find and work with.

Parameters	Model registry	Experiment tracking
Purpose	To store trained, production-ready, and retired models in a central repository.	To track experiment runs of different parameter configurations and combinations.
Priority	To make sure models are discoverable and can be accessed by any user or system.	To make sure experiments are easier to manage and collaborate on.
Integration	Integrates with the experiment tracking system to register models from successful experiments including the model and experiment metadata.	Integrates with the training pipeline to perform experiment runs and track experiment details including the dataset version and metadata.
MLOps	A crucial piece of MLOps and production models.	Most useful in the model development phase, has an indirect impact on operationalizing the model.

Model registry key components



Model registry key features and functionalities

- Acts as a centralized storage for effective collaboration
 - Model registry provides a central storage unit that holds models (including model artifacts) for easy retrieval by an application (or service)
 - Without the model registry, the model artifacts would be stored in files that are difficult to track and saved to whatever source code repository is established
 - The centralized storage also enables data teams to have a single view of the status of all models, making collaboration easier

Bridges the gap between experiment and production activities

- Model registry acts as a glue between ML experimentation and Operations, enabling model development, software development, and operational teams to collaborate.



Model registry key features and functionalities(2)

Providing a central UI

- Model registry provides teams with visibility over their models
- With a central interface, teams can:
 - Search for models,
 - View the status of models (if they are being staged, deployed, or retired),
 - Approve or disapprove models across different stages,
 - And view the necessary documentation.
- This makes model discovery easier for everyone on the team.
- If a model needs to be deployed, the operations teams can easily:
 - Search for it,
 - Look up the validation results and other metrics,
 - Package the model (if needed),
 - And move it from the staging environment to the production environment.
- This improves the way cross-functional teams collaborate on ML projects.

Model registry key features and functionalities(3)

Enables model versioning by tracking different versions of a model

- Model registry enables model versioning by tracking the different versions of a model as they are developed and improved
- This allows to compare different versions of the model, track its performance, and select the best version for deployment
- Here's how it typically works:
 - Model Registration: When a new model is developed or trained, it is registered in the model registry.
 - Version Control: The model registry maintains a history of all registered models and their versions.
 - Model Comparison: The model registry allows users to compare performance metrics, model architectures, hyperparameters, and other relevant information in different versions of a model.
 - Model Tracking: As new versions of the model are developed or trained, they are registered in the model registry as well, incrementing the version number.
 - Retention and Archiving: The model registry typically retains older versions of the models, ensuring a complete history and traceability.
 - By enabling model versioning, the model registry ensures that different iterations of a model can be stored, tracked, compared, and accessed conveniently.

Model registry key features and functionalities(4)

Integrates with experiment management systems or training pipelines

- Model registry integrates with systems that output the trained model
- The trained models could be the raw artifacts (model weights, configuration, and metadata) or models that have been serialized into a file (e.g., an ONNX file) for compatibility with the production environment or containerized (using Docker) to be exported to the production environment.
- Model registries:
 - Register the model,
 - Assign a version to it,
 - Note the version of the dataset the model was trained on,
 - Add annotations and tags,
 - Retrieve the parameters, validation results (including metrics and visualizations), and other relevant model metadata on the model from the experiment management system.
- To make collaboration easier, the registry also includes details such as:
 - The model owner or developer,
 - Experiment run id the model was trained under,
 - Versioned model source code,
 - Environment runtime dependencies used to train the model (and the versions),
 - Comments and model change history,
 - And the model documentation.

Model registry key features and functionalities(5)

Integrates with the staging environment for trained models

- Model Registry provides the functionality for integrating with the staging environment for running all types of checks and balances on the model
 - can include integration testing (with other applications) and other QA tests before the model can be promoted to the production environment
- In the staging environment, the model reviewers should also be able to perform fairness checks on the model to ensure it:
 - Outputs explainable results,
 - Complies with regulatory requirements,
 - And provides useful business benefits.

Model registry key features and functionalities(6)

Integrate with model delivery (CI/CD) tools and services for automation

- Automation is a critical part of building any scalable software
 - In machine learning, building automated pipelines will allow to spend more time building new products rather than maintaining old models
- A Model registry integrates with pipeline automation tools and provides custom APIs that can allow to plug into custom workflow tools
 - For example, using webhooks to trigger downstream actions based on predefined events in the registry.
- Should also be able to configure model promotion schemes through different environments like development (training), staging (testing), and production (serving)
- Performance is a crucial requirement for building automated pipelines
 - Model registries should be highly available for automated jobs that are event or schedule-based to enable continuous training and delivery of the model

Model registry key features and functionalities(7)

- Provides an interface for downstream systems to consume models
 - Model registries provide interfaces that enable downstream services to consume the model through API integration
 - The integration can also track offline and online evaluation metrics for the models
 - makes it easy to build an automated setup with CI/CD/CT with ML pipelines
 - downstream service could be either a model user, an automated job, or a REST serving that can consume the most stable—or any—version of the model
- Integrate with model deployment tools
 - Eventually, models have to be deployed, and the more efficient the deployment process, the better.
 - Model Registries:
 - Integrate with downstream services and REST serving services that can consume the model and serve it in the production environment.
 - Collect real-time (or aggregated) metrics on the production model to log performance details of the model. This will be helpful for comparison between models (deployed and staged), as well as auditing the production model for review



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

ML Model Registry - How?

Pravin Y Pawar

Adapted [from ML Model Registry](#)
By Stephen Oladele

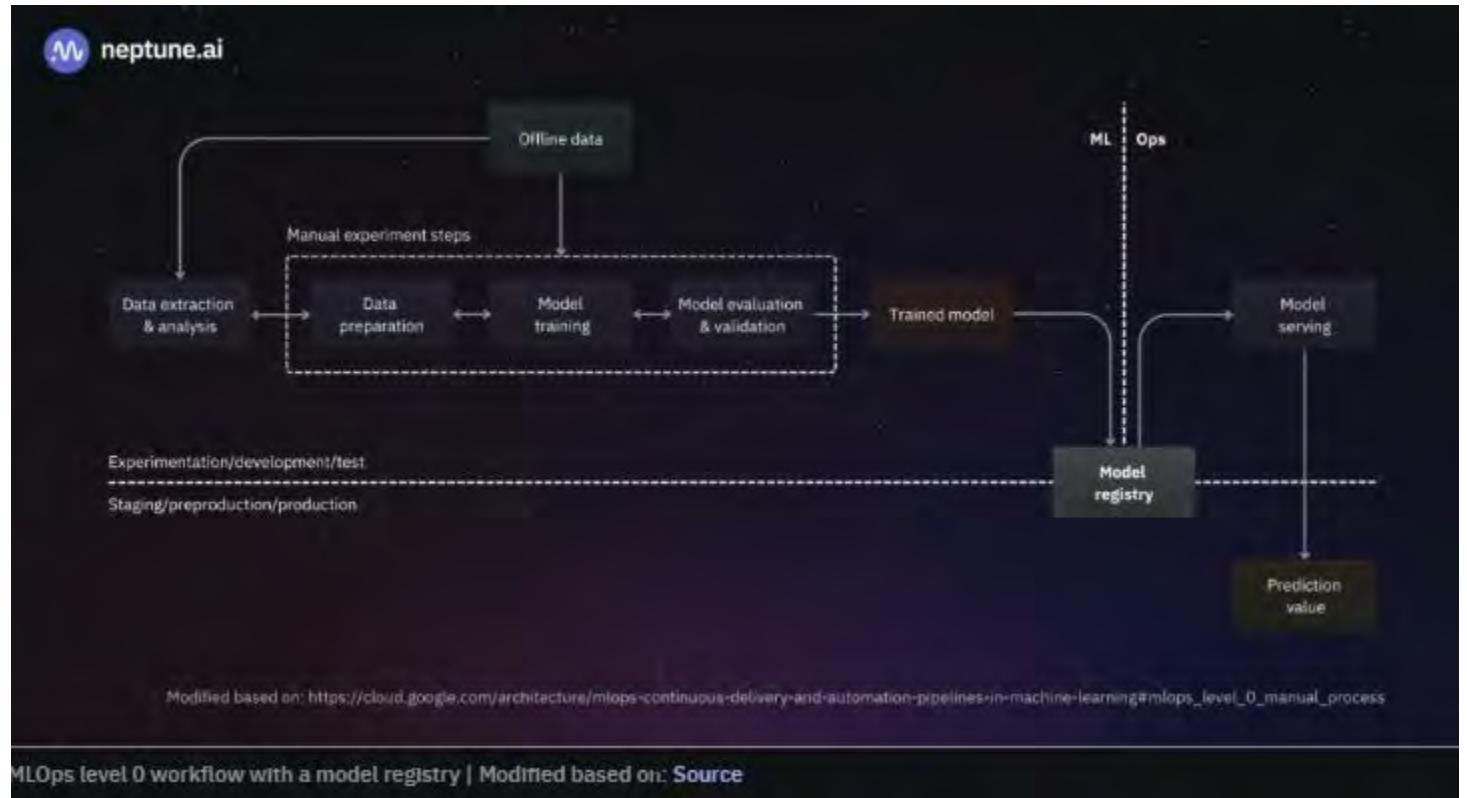
Where does a model registry fit in the MLOps stack?

- If want to run machine learning projects efficiently and at scale,
 - most likely need to add a model registry to MLOps stack.
- Depending on what level of implementation you are in your MLOps stack,
 - needs and requirements for a model registry would differ.
- Where does it fit?
 - the model registry sits between machine learning development and deployment.

Where does a model registry fit in the MLOps stack?(2)

Model registry in MLOps level 0

- The output from the experimentation step is fed into the model registry.
 - involves a manual process where the data scientist prepares the model artifact and metadata and could also package them (serialization, containerization) before registering them
- The operations team can push the packaged model to the staging environment
 - for testing before deploying it to a prediction service engine that can integrate with other applications.



Where does a model registry fit in the MLOps stack? (3)

Model registry in MLOps level 1

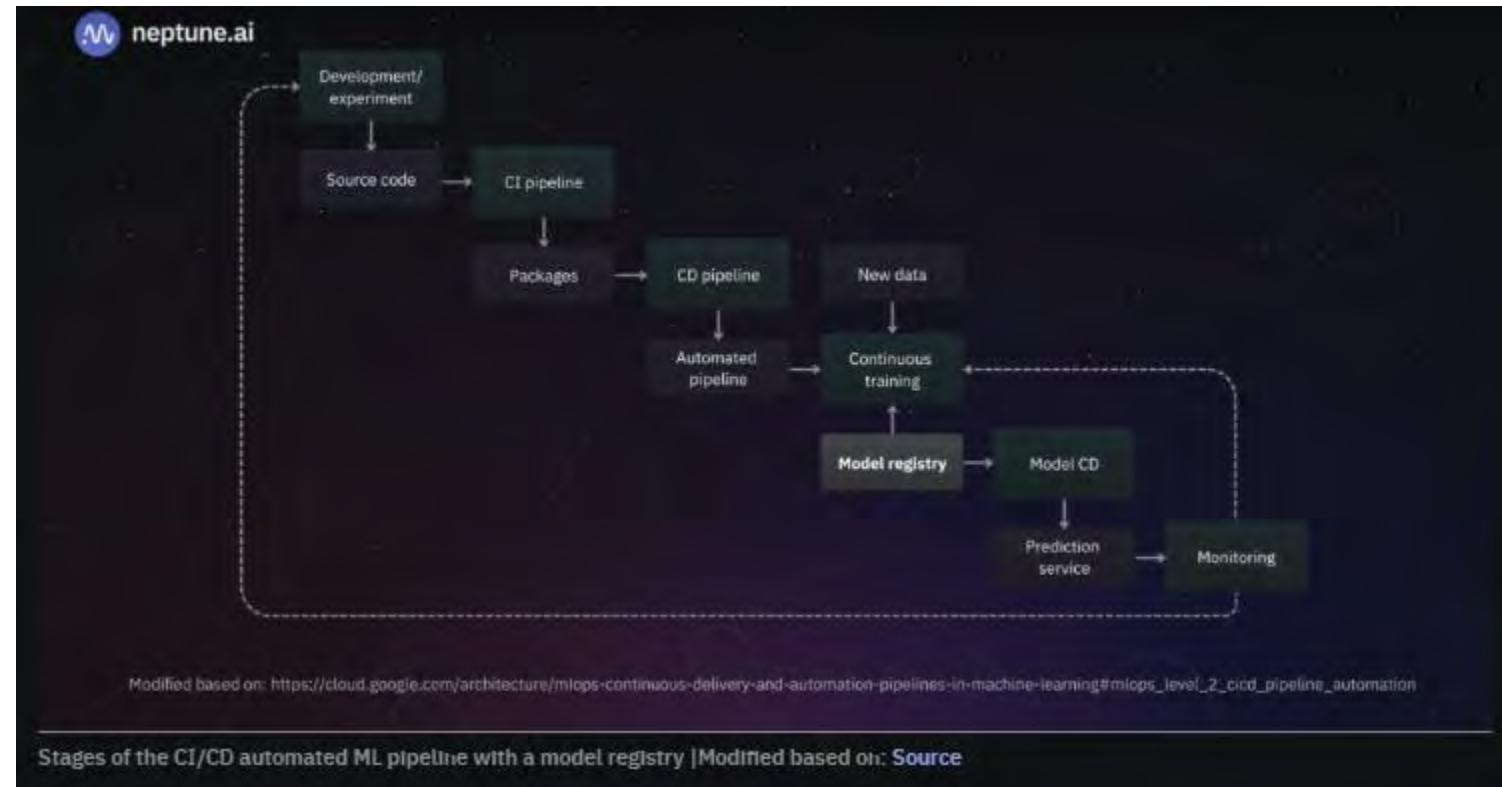
- As opposed to level 0 (where the workflow is a manual process), the goal of the workflow in level 1 is to perform continuous training of the model by automating the ML pipeline. This is one process a model registry enables well because of its ability to integrate with the pipeline.
- At this level, the entire pipeline is deployed, and when models are trained on the provided dataset, the output (trained model and metadata) is fed into the model registry where it can be staged, and if it passes the necessary tests and checks, it can be fed to the continuous delivery pipeline for release.



Where does a model registry fit in the MLOps stack?(4)

Model registry in MLOps level 2

- The role of the model registry in level 2 of the MLOps workflow is also the same as that of level 1—the automated pipeline delivers the trained model to the model registry where it is staged, may be passed through QA checks, and sent to the continuous delivery pipeline:
- The model registry serves as a crucial component in any automated pipeline because event triggers can be integrated with it to promote models with good metrics upon re-training on fresh data or archive models.



Setting up ML model registry – build, maintain or buy?

- Setting up a model registry for MLOps workflow will require you to decide on either
 - Building one,
 - Maintaining one,
 - or Buying one

Setting up ML model registry – build, maintain or buy?(2)

Building a model registry solution

- Like any software solution, if you understand the key functionalities and requirements, you can build a system yourself! This is the case with a model registry.
- May want to set up the following:
 - Object storage for models and artifacts
 - Database for logging model details
 - API integration for both receiving models, promoting models across various environments, and collecting model information from the different environments
 - User interface (UI) for ML teams to interact with a visual workflow
- While building the solution yourself might seem ideal, you should consider the following factors:
 - Incentive: What's the incentive to build out your solution? Is it for customization or for owning a proprietary license to the solution?
 - Human resources: Do you have the talents and skills to build out your solution?
 - Time: How long would it take you to build out a solution and is it worth the wait?
 - Operations: When the solution is eventually built out, who would maintain its operations?
 - Cost: What would it cost you to build a solution, including the maintenance of the solution?

Setting up ML model registry – build, maintain or buy?(3)

Maintaining a self-hosted model registry

- Another option to consider is to maintain an existing solution yourself
 - solution has been built out already, but might have to manage some features, such as object storage and database
 - most of these existing solutions are open-source solutions
- The following are the factors to consider:
 - Type of solution: Are you going to opt for an open-source solution with no license cost or a closed-source solution with license cost?
 - Operations: Who is going to manage the solution? Does the solution support consistent maintenance and software updates?
 - Cost: What is the cost of operating the solution in terms of the infrastructure to host it and the running cost?
 - Features: What features have already been implemented, and what features do you have to build and manage yourself? Is it worth adopting compared to building out your solution?
 - Support: What type of support is available in case things break during operations? Is there a community or dedicated customer support channel? For open-source solutions, while you might have a community, you will likely lack the necessary developer support required to fix things.
 - Accessibility: How easy is it to get started with the solution? Is the documentation comprehensive enough? Can everyone from the model reviewers to the model developers and software engineers intuitively use the solution?

Setting up ML model registry – build, maintain or buy?(4)

Purchase the license to a fully-managed solution

- The final option to consider is subscribing to a fully managed solution where the operations and management of the registry are handled by the solution vendor
 - do not have to worry about building or maintaining a solution
 - just have to ensure systems and services can integrate with the registry
- Here are the factors to consider:
- Industry type: What type of industry is the model built for? What sensitive information have the models learned? Are there data privacy compliance measures? Is the model only allowed to stay on-premise?
- Features: Are the key features and functionalities of any model registry available in this solution? What extra features are available, and how relevant are they to your workflow?
- Cost: What's the cost of purchasing a license, and do the features justify the cost?
- Security: How secure is the platform hosting the solution? Is it resistant to third-party attacks?
- Performance: Is the registry highly performant? For situations where models are too large, can the registry provide models for services to consume at low latency?
- Availability: What's the uptime of the solution, and does it meet your required service level agreement (SLA)?
- Support: What level of support is available in case things go south?
- Accessibility: How easy is it to get started with the solution? Are the documentation and learning support decent enough? What's the learning curve in terms of usage?

ML model registry solutions out there

The most popular ones available out there that offer a host of Model Registry features.

Model Registry Features	Azure ML Studio	Neptune.ai	MLFlow	Verta.ai	AWS Sagemaker
Model Versioning	✓	✓	✓	✓	✗
CI/CD/CT compatibility	✓	✗	✓	✓	✗
Model packaging	✗	✗	✓	✓	✓
Model searching	✓	✓	✓	✓	✓
Access control, model review, and promoting models	✗	✓	✓	✓	✓
Model lineage and evaluation history	✓	✗	✗	✓	✓



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Metadata

Pravin Y Pawar

Adapted from Reliable Machine Learning
By Cathy Chen

Dataset Metadata

Metadata about features and labels

- Dataset provenance
 - Where did the data come from?
 - Depending on the source of data, might have a lookup table of logs from various systems, a key for an external data provider with data about when we downloaded the data, or even a reference to the code that generated the data
- Dataset location
 - For some datasets, will store raw, unprocessed data
 - should store a reference to where dataset is kept, as well as perhaps information about where we got it from
 - Some data is created for on an ongoing basis, such as logs from systems - should store the log or datastore reference where
 - that data is stored, or where we are permitted to read from it.
- Dataset responsible person or team
 - Should track which person or team is responsible for the dataset
 - In general, this is the team that chose to download or create the dataset, or the team that owns the system that produces the data
- Dataset creation date or version date
 - Often useful to know the first date a particular dataset was used
- Dataset use restrictions
 - Many datasets have restrictions on their use, either because of licensing or governance constraints
 - should document that in the metadata system for easy analysis and compliance later

Feature Metadata

Track of metadata about feature definitions

- Feature version definition
 - a reference to code or another durable description to what data the feature reads and how it processes the data to create the feature
 - should be updated for every updated version of the feature definition
 - versioning these definitions (and restricting the versions in use) will make the resulting codebase more predictable and maintainable
- Feature definition responsible person or team
 - There are two good use cases for storing this information:
 - figuring out what a feature is for
 - and finding someone who can help resolve an incident when the feature might be at fault
 - In both cases, it is useful to store authorship or maintainer information about the feature
- Feature definition creation date or current version date
 - useful to get a change history of when a feature was most recently updated and when it was originally created
- Feature use restrictions
 - important but trickier to store
 - Features may be restricted from use in some contexts.
 - For example, it may be illegal to use a particular feature in some jurisdictions.
 - Age and gender may be a reasonable predictor of automobile insurance risk models, but insurance is highly regulated
 - may not be permitted to take those fields into account

Label Metadata

Intended to help with the maintenance and development of the labeling system itself, but might also be used by the training system as it uses the labels

- Label definition version
 - Switching to metadata specific to labels and analogously to features,
 - must store the version of any label definitions to understand which labeling instructions the labels were made with
- Label set version
 - Changes to the labels may occur because of incorrect labels getting corrected or new labels being added
 - If the dataset is being used for comparison with an older model, using an older version of the labels may be desirable
- Label source
 - Although not typically needed for training, it is sometimes necessary to know the source of each label in a dataset
 - may be the source that particular label was licensed from,
 - the human who produced the label(along with any QA that was applied to the label),
 - or the algorithm that produced the label if an automated labeling approach was used
- Label confidence
 - Depending on how the labels are produced, might have different estimates of the confidence of correctness for different labels
 - Users of these labels might choose different thresholds to decide which labels to use in training their models

Pipeline Metadata

- Won't spend as much time on: metadata about the pipeline processes themselves
 - data about the intermediate artifacts we have,
 - which pipeline runs they came from,
 - and which binaries produced the
- Type of metadata is produced automatically by some ML training systems
 - for example, ML Metadata (MLMD) is automatically included in TensorFlow Extended (TFX),
 - which uses it to store artifacts about training runs
- Such systems are either integrated into training systems or are somewhat difficult to implement later!

Metadata Systems Overview

Feature systems and labeling systems both benefit from efficient tracking of metadata

- A metadata system is designed to keep track of what we're doing
 - In the case of features and labels, it should minimally keep track of the feature definitions and the versions used in each model's definitions and trained models
- Most organizations start building their data sciences and ML infrastructure without a solid metadata system,
 - only to regret it later!
- The next most common approach is to build several metadata systems, each targeted at solving a particular problem
 - make one for tracking feature definitions and mappings to feature stores
 - need a system for mapping model definitions to trained models, along with data about the engineers or teams responsible for those models
 - need a model serving system is also going to need to keep track of trained model versions, when they were put into production
 - need a model quality or fairness evaluation systems will need to be read from all of these systems in order to identify and track the likely contributing causes of changes in model quality or violations of our proposed fairness metrics

Choices for a metadata system

One system

- Build one system to track metadata from all of these sources
 - makes it simple to make correlations across multiple subsystems, simplifying analysis and reporting
- Such a large system is difficult to get right from a data schema perspective
 - will be constantly adding columns when we discover data we would like to keep track of (and backfilling those columns on existing data)
 - be difficult to stabilize such a system and make it reliable
- From a systems design perspective, should ensure that our metadata systems are never in the live path of either model training or model serving
 - But it's difficult to imagine how feature engineering or labeling can take place without the metadata system being functional
 - can still cause production problems for our humans working on those tasks
- If the needs are simple and well understood, prefer a single system.

Choices for a metadata system (2)

Multiple systems (that work together)

- Can build separate metadata systems for each task we identify
 - Perhaps would have one for features and labels, one for training, one for serving and one for quality monitoring
- Decoupling the system provides the standard advantages and costs that decoupling always does
 - allows to develop each system separately without concern for the others, making them nimbler and simpler to modify and extend
 - an outage of one metadata system has limited production impact on others
 - The cost, though, is the added difficulty of analysis and reporting across those systems
 - Will have processes that need to join data across the features, labeling, training, and serving systems
- The area is rapidly developing and teams expect to continue extending what they track and how they work, multiple systems will simplify the development over time.

Most Ignored!

- More generally, metadata systems are often overlooked or deprioritized
 - They should not be
 - They are one of the most effective and direct contributors to productive use of the data in an ML system
 - Metadata unlocks value and should be prioritized



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Pravin Y Pawar

Adapted from
[ML Metadata Store: What It Is, Why It Matters, and How to Implement It](#)
by Jakub Czakon

Need for metadata

But the problems people have with storing and managing ML model metadata are different.

- For some, it is messy experimentation that is the issue
- Others have already deployed the first models to production, but they don't know how those models were created or which data was used
- Some people already have many models in production, but
 - orchestrating model A/B testing,
 - switching challengers and champions,
 - or triggering, testing, and monitoring re-training pipelines is not great.
- ML metadata store can help with all of those things and then some others as well.

What is ML metadata anyway?

- When you do machine learning, there is always a model involved. It is just what machine learning is.
- ML Model could be a classic, supervised model like a lightGBM classifier, a reinforcement learning agent, a bayesian optimization algorithm, or anything else really.
 - But it will take some data, run it through some numbers and output a decision.
 - ... and it takes a lot of work to deliver it into production.
- One has to:
 - train,
 - tune,
 - debug,
 - evaluate,
 - explain,
 - and compare it to baseline.
- If model makes it past the research phase, it also has to :
 - package,
 - deploy,
 - monitor,
 - and re-train it.

What is ML metadata anyway? (2)

- A lot of steps!
 - Don't go through those steps linearly, and you don't go through them once.
 - Takes a lot of iterations of fixing data, fixing model, fixing preprocessing, and all that good stuff.
 - **Each of those steps produces meta-information about the model, or as many people call it, ML model metadata.**
- Those could be:
 - training parameters,
 - evaluation metrics,
 - prediction examples,
 - dataset versions,
 - testing pipeline outputs,
 - references to model weights files,
 - and other model-related things.
- This information helps you know your models.
 - Where the particular version of the model is and quickly rollback to the previous version
 - How the model was built and who created it
 - Which data/parameters/code it used at training
 - How does the new experiment or model version compare to the baseline or previous production models
 - How did it perform at various evaluation stages
 - And when your models are important to someone, when they touch users or help clients make decisions, you better know a lot about them.

Metadata about experiments and model training runs

- During experimentation, you usually care about debugging, visualizing, monitoring your model training to get to the best model.
- To do that, it is a good practice to log anything that happens during the ML run, including:
 - data version: reference to the dataset, md5 hash, dataset sample to know which data was used to train the model
 - environment configuration: requirements.txt, conda.yml, Dockerfile, Makefile to know how to recreate the environment where the model was trained
 - code version: git SHA of a commit or an actual snapshot of code to know what code was used to build a model
 - hyperparameters: configuration of the feature preprocessing steps of the pipeline, model training, and inference to reproduce the process if needed
 - training metrics and losses: both single values and learning curves to see whether it makes sense to continue training
 - record of hardware metrics: CPU, GPU, TPU, Memory to see how much your model consumes during training/inference
 - evaluation and test metrics: f2, acc, roc on test and validation set to know how your model performs
 - performance visualizations: ROC curve, Confusion matrix, PR curve to understand the errors deeply
 - model predictions: to see the actual predictions and understand model performance beyond metrics
 - ...and about a million other things that are specific to your domain

Metadata about artifacts

- Apart from experiments and model training runs, there is one more concept used in ML projects: artifact
 - input or output of those runs can be used in many runs across the project
 - Artifacts can change during the project, and typically have many versions of the same artifact at some point in your ML lifecycle
 - Artifacts could be datasets, models, predictions, and other file-like objects.
- For artifacts you may want to log:
 - Reference: Paths to the dataset or model (s3 bucket, filesystem)
 - Version: Dataset or model md5 hash to let you quickly see the diff
 - Preview: Dataset/prediction preview (head of the table, snapshot of the image folder) to see what this dataset is about
 - Description: additional info about the artifact that will help you understand what it is. For example, you may want to log column names for a tabular dataset artifact
 - Authors: who created modified this artifact and when
 - And many other things that may be specific to your project as the size of the dataset, type of the ML model and others

Metadata about trained models

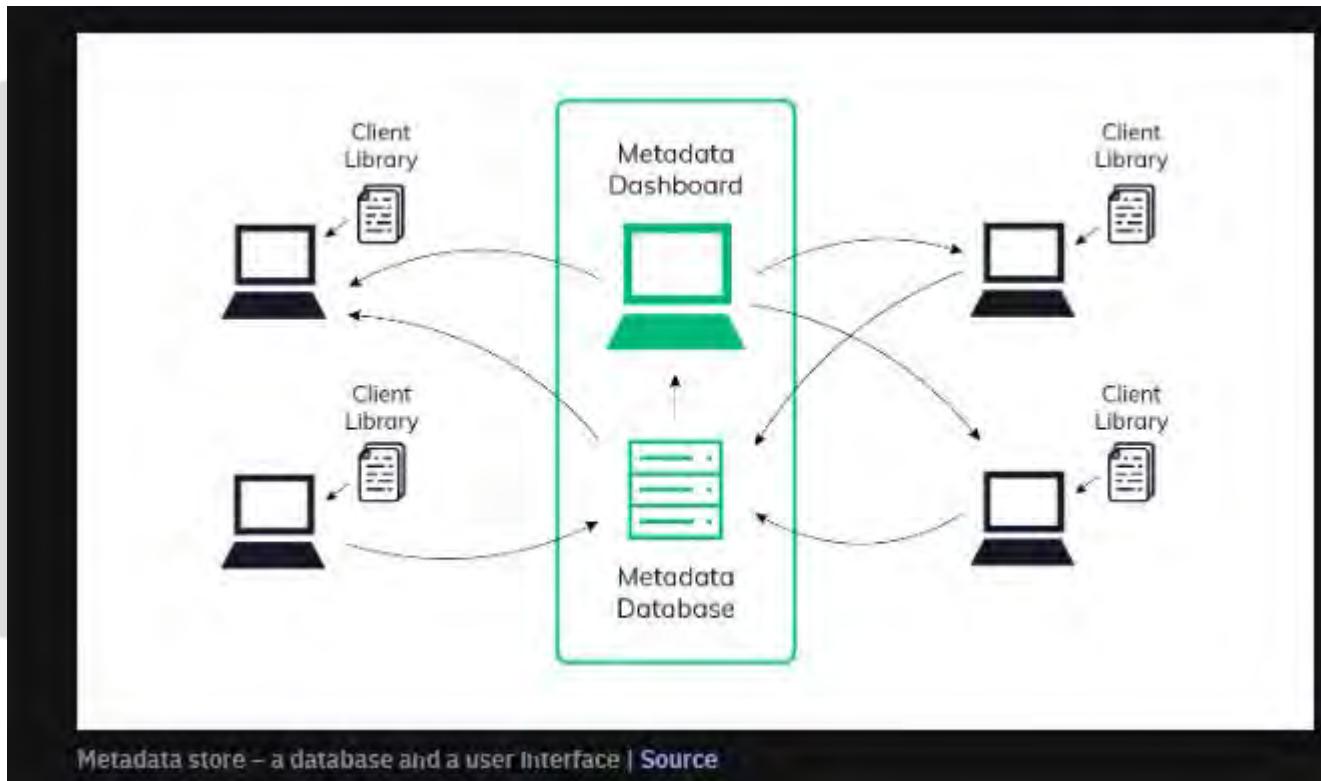
- Trained models are such an important type of artifact in ML projects
 - Once your model is trained and ready for production, needs change from debugging and visualization to knowing how to deploy a model package, version it, and monitor the performance on prod.
- So the ML metadata may want to log are:
 - Model package: Model binary or location to your model asset
 - Model version: code, dataset, hyperparameters, environment versions
 - Evaluation records: History record of all the evaluations on test/validation that happened over time
 - Experiment versions: Links to recorded model training (and re-training) runs and other experiments associated with this model version
 - Model creator/maintainer: who build this model, and who should you ask if/when things go wrong
 - Downstream datasets/artifacts: references of datasets, models, and other assets used downstream to build a model. This can be essential in some orgs for compliance.
 - Drift-related metrics: Data drift, concept drift, performance drift, for all the “live” production
 - Hardware monitoring: CPU/GPU/TPU/Memory that is consumed in production.
 - And anything else that will let you sleep at night while your model is sending predictions to the world

Metadata about pipeline

- You may be at a point where your ML models are trained in a pipeline that is triggered automatically:
 - When the performance drops below certain thresholds
 - When new labeled data arrives in the database
 - When a feature branch is merged to develop
 - Or simply every week
- Likely have some CI/CD workflow connected to a pipeline and orchestration tool like Airflow, Kubeflow, or Kedro
 - every trigger starts the execution of a computation DAG (directed acyclic graph) where every node produces metadata
- Need for the metadata is a bit different than for experiments or models
 - This metadata is needed to compute the pipeline (DAG) efficiently:
 - Input and output steps: information about what goes into a node and what goes out from a node and whether all the input steps are completed
 - Cached outputs: references to intermediate results from a pipeline so that you can resume calculations from a certain point in the graph

What is an ML metadata store?

- ML metadata store is a “store” for ML model-related metadata
 - place where one can get anything one need when it comes to any and every ML model you build and deploy.
- More specifically, ML metadata store lets you:
 - log,
 - store,
 - display,
 - monitor,
 - compare,
 - organize,
 - filter,
 - and query all model-related metadata.
- In short, it gives a single place to manage all the ML metadata about experiments, artifacts, models, and pipelines we have listed in the previous section.
- You can think of it as a database and a user interface built specifically to manage ML model metadata
- It typically comes with an API or an SDK (client library) to simplify logging and querying ML metadata.



Repository vs Registry vs Store

- In the context of ML metadata,
 - all of those things are basically databases created to store metadata that have slight functional differences
- A metadata repository
 - place where store metadata objects along with all the relationships between those objects
 - can use GitHub as a repository where save all the evaluation metrics as a file created via post-commit hook or log parameters and losses during training to an experiment tracking tool which in this context is actually a repository
- A metadata registry
 - place where you “checkpoint” the important metadata
 - register something you care about and want to easily find and access it later
 - A registry is always for something specific. There are no general registries
 - For example, you may have a model registry that lists all the production models with references to the ML metadata repository where the actual model-related metadata is.
- A metadata store
 - place where you go “shopping” for metadata
 - For ML models, it is a central place where you can find all the model, experiment, artifact, and pipeline-related metadata
 - The more you need to come to the “shop”, search metadata “products”, compare them and “buy” them, the more it is a store rather than a repository.

Two flavors of ML metadata store

Pipeline-first vs model-first ML metadata store

- As your ML organization matures, you get to a point when training models happen in some automated, orchestrated pipelines
 - At that moment, running experiments, training, and re-training production models is always associated with executing a pipeline.
- ML metadata store can:
 - Treat pipelines as first-class objects and associate resulting models and experiments with them
 - Treat models and experiments as first-class objects and treat pipelines as a mode of execution
 - Depending on what you put in a center, your ML metadata store will do slightly different things
- If you focus on the process, the model building pipeline, you get something like MLMD, which powers pipeline and orchestration tools like Kubeflow and TensorFlow Extended (TFX).
- If you focus on the output, the model, you get something like MLflow or Neptune.

How do you set up an ML metadata management system?

Build vs maintain vs buy

- It's an old dilemma!
 - People who work in software face it many times in their careers.
- To make a decision, you should:
 - understand what problem you are actually solving
 - see if there are tools that can solve this problem before building it
 - assess how much time it will take to build, maintain, and customize it
 - don't underestimate the amount of effort that goes into it

How do you set up an ML metadata management system?(2)

Build a system yourself

Pros	Cons
No licence cost	You have to implement it
You can create it exactly for your use case	You have to set up and maintain the infrastructure
You can improve/change it however you like	You have to document it for other people
	You have to fix bugs
	You have to implement improvements and integrations with other tools
	When software doesn't work it is your fault

How do you set up an ML metadata management system?(3)

Maintain open source

Pros	Cons
No licence cost	You have to set up and maintain the infrastructure
You don't have to implement it	You have to adjust it to your needs
Documentation and community support is already there	You have to count on community support
Other people are creating improvements that you may care about	You can push for new features but may have to implement them yourself
	When software doesn't work it is your fault
	Project may get abandoned by the creators/maintainers/community

How do you set up an ML metadata management system?(4)

Buy a solution

Pros	Cons
You don't have to implement it	Licence fee
Documentation and support is provided by the vendor	You may not be able to fix bugs/create improvements yourself and the vendor may be slow to fix them
Infrastructure is already there (for hosted options)	Company may go out of the market
When you need features you can request them	
When software doesn't work it is their fault (and they have to fix it)	



Thank You!

In our next session:



BITS Pilani
Pilani Campus

CSIW ZG514: DATA WAREHOUSING

Contact Session - 13

Faculty:

IMP Note to Self



**Start
Recording**



CSIW ZG514: DATA WAREHOUSING

CS 13 : Support for Data Warehousing in RDBMS/SQL

IMP Note to Students

- It is important to know that just login to the session does not guarantee the attendance.
- Once you join the session, continue till the end to consider you as present in the class.
- IMPORTANTLY, you need to make the class more interactive by responding to Professors queries in the session.
- Whenever Professor calls your number / name ,you need to respond, otherwise it will be considered as ABSENT.

Coverage

Time	Type	Description	Content Reference
Pre-CS	RL 9.1.1	Aggregation	T1 - Ch 1 R4 - Ch 2, Ch 3, Ch 18
	RL 9.1.2	Partitioning & Materialized Views	
	RL 9.1.3	Bitmap Indexes & Dimensions	
	RL 9.2.1	SQL - Rollup & Cube	
	RL 9.2.2	SQL - Window Queries & Top N Queries	
During CS	CS 13	Commonality between DW and RDBMS	
		SQL extensions for analytics	
Post-CS	SS 13	Self-Study (R4, Chapters 2, 3, 18)	
Lab Reference			

Aggregation

- An aggregate is a type of summary used in dimensional models of data warehouses to shorten the time it takes to provide answers to typical queries on large sets of data.
- The reason why aggregates can make such a dramatic increase in the performance of a data warehouse is the reduction of the number of rows to be accessed when responding to a query.
- In its simplest form, an aggregate is a simple summary table that can be derived by performing a Group by SQL query.
- A more common use of aggregates is to take a dimension and change its granularity.
- When changing the granularity of the dimension the fact table has to be partially summarized to fit the new grain of the new dimension, thus creating new dimensional and fact tables, to fit this new level of grain.

Aggregate Navigation

- The aggregate navigator is an important part of the data warehouse architecture.
- It maintains its own inventory of aggregates, their size, and availability, and rewrites queries as appropriate.
- In doing so, it shields users from the table selection process and allows all queries and reports to be expressed in terms of the base schema.
- Without an aggregate navigator, aggregates damage the understandability of the schema, threaten its acceptance by end users, weaken its efficiency, and introduce maintenance issues.
- If aggregates are queried at a time when they are not in synch with the base tables, they may also provide wrong results

Aggregate Navigation

- Having aggregate data in the dimensional model makes the environment more complex.
- To make this extra complexity transparent to the user, functionality known as aggregate navigation is used to query the dimensional and fact tables with the correct grain level.
- The aggregate navigation essentially examines the query to see if it can be answered using a smaller, aggregate table.
- Implementations of aggregate navigators can be found in a range of technologies:
 - OLAP engines
 - Materialized views
 - Relational OLAP (ROLAP) services
 - BI application servers or query tools

Setting up Aggregate Navigation

- When you create a logical table source for an aggregate fact table, you should create corresponding logical dimension table sources at the same levels of aggregation.
- You need to have at least one logical dimension table source for each level of aggregation. If the sources at each level already exist, you do not need to create new ones.
- For example, you might have a monthly sales fact table containing a precomputed sum of the revenue for each product in each store during each month.
- You need to have the following three other dimension sources, one for each of the logical dimension tables referenced in the example-

Setting up Aggregate Navigation

A source for the Product logical table with one of the following content specifications:

- By logical level: ProductDimension.ProductLevel
- By column: Product.Product_Name

A source for the Store logical table with one of the following content specifications:

- By logical level: StoreDimension.StoreLevel
- By column: Store.Store_Name

A source for the Time logical table with one of the following content specifications:

- By logical level: TimeDimension.MonthLevel
- By column: Time.Month

Partitioning

- Breaking data into several physical units that can be handled separately.
- Partitioning is key to effective implementation of a warehouse.
- Partitioning increases the speed of data access and application processing, which results in successful data warehouses that are not prohibitively expensive.
- Partitioning is supported in most RDBMSs
- Even if it is not supported, one can create them using the concept of views by-
 - Manually move data from the table to be partitioned to its partitions (tables)
 - Create a view using union of partitions, giving an illusion that the original table still exists

Types of Partitioning

Oracle offers four partitioning methods:

- Range Partitioning
- Hash Partitioning
- List Partitioning
- Composite Partitioning

Each partitioning method has different advantages and design considerations.

Thus, each method is more appropriate for a particular situation

Range Partitioning

- Range partitioning maps data to partitions based on ranges of partition key values that you establish for each partition.
- It is the most common type of partitioning and is often used with dates.
- For example, you might want to partition sales data into monthly partitions.
- Ranges should be contiguous but not overlapping, and are defined using the VALUES LESS THAN operator.
- Range partitioning is defined by the partitioning specification for a table or index in PARTITION BY RANGE(*column_list*).
 - where *column_list* is an ordered list of columns that determines the partition to which a row or an index entry belongs.

Range Partitioning- Example 1

- Suppose that you are creating a table such as the following to hold personnel records for a chain of 20 video stores, numbered 1 through 20.

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
    store_id INT NOT NULL
);
```



Use *store_id* column for partition

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
    store_id INT NOT NULL
)
PARTITION BY RANGE (store_id) (
    PARTITION p0 VALUES LESS THAN (6),
    PARTITION p1 VALUES LESS THAN (11),
    PARTITION p2 VALUES LESS THAN (16),
    PARTITION p3 VALUES LESS THAN (21)
);
```

Range Partitioning- Example 2

Partition the table by **RANGE**, and for the partitioning expression, employ a function operating on a DATE, TIME, or DATETIME column and returning an integer value, as shown here:

```
CREATE TABLE members (
    firstname VARCHAR(25) NOT NULL,
    lastname VARCHAR(25) NOT NULL,
    username VARCHAR(16) NOT NULL,
    email VARCHAR(35),
    joined DATE NOT NULL
)
PARTITION BY RANGE( YEAR(joined) ) (
    PARTITION p0 VALUES LESS THAN (1960),
    PARTITION p1 VALUES LESS THAN (1970),
    PARTITION p2 VALUES LESS THAN (1980),
    PARTITION p3 VALUES LESS THAN (1990),
    PARTITION p4 VALUES LESS THAN MAXVALUE
);
```

Note:

MAXVALUE literal can be defined for the highest partition

Hash Partitioning

- Hash partitioning maps data to partitions based on a hashing algorithm that applies to a partitioning key that you identify.
- The hashing algorithm evenly distributes rows among partitions, giving partitions approximately the same size.
- Hash partitioning is the ideal method for distributing data evenly across devices.
- Hash partitioning is also an easy-to-use alternative to range partitioning, especially when the data to be partitioned is not historical.
- To partition a table using HASH partitioning, it is necessary to append to the CREATE TABLE statement a PARTITION BY HASH (expr) clause, where expr is an expression that returns an integer.

Hash Partitioning- Example

The following statement creates a table that uses hashing on the `store_id` column and is divided into 4 partitions:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY HASH(store_id)
PARTITIONS 4;
```

If you do not include a `PARTITIONS` clause, the number of partitions defaults to 1.

Using the `PARTITIONS` keyword without a number following it results in a syntax error.

List Partitioning

- As similar to partitioning by RANGE, each partition must be explicitly defined.
- The chief difference between the two types of partitioning is that-
 - ✓ In list partitioning, each partition is defined and selected based on the membership of a column value in one of a set of value lists, rather than in one of a set of contiguous ranges of values.
- This is done by using PARTITION BY LIST(expr) where expr is a column value or an expression based on a column value and returning an integer value, and then defining each partition by means of a VALUES IN (*value_list*), where *value_list* is a comma-separated list of integers.

List Partitioning- Example 1

- The advantage of list partitioning is that you can group and organize unordered and unrelated sets of data in a natural way.
- The following example creates a list partitioned table grouping states according to their sales regions:

```
CREATE TABLE sales_list
(salesman_id NUMBER(5),
salesman_name VARCHAR2(30),
sales_state VARCHAR2(20),
sales_amount NUMBER(10),
sales_date DATE)
PARTITION BY LIST(sales_state)
(PARTITION sales_west VALUES('California', 'Hawaii') COMPRESS,
PARTITION sales_east VALUES('New York', 'Virginia', 'Florida'),
PARTITION sales_central VALUES('Texas', 'Illinois'));
```

List Partitioning- Example 2

- Suppose that there are 20 video stores distributed among 4 franchises as shown in the following table.

Region	Store ID Numbers
North	3, 5, 6, 9, 17
East	1, 2, 10, 11, 19, 20
West	4, 12, 13, 14, 18
Central	7, 8, 15, 16



```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY LIST(store_id) (
    PARTITION pNorth VALUES IN (3,5,6,9,17),
    PARTITION pEast VALUES IN (1,2,10,11,19,20),
    PARTITION pWest VALUES IN (4,12,13,14,18),
    PARTITION pCentral VALUES IN (7,8,15,16)
);
```

Key Partitioning- Example

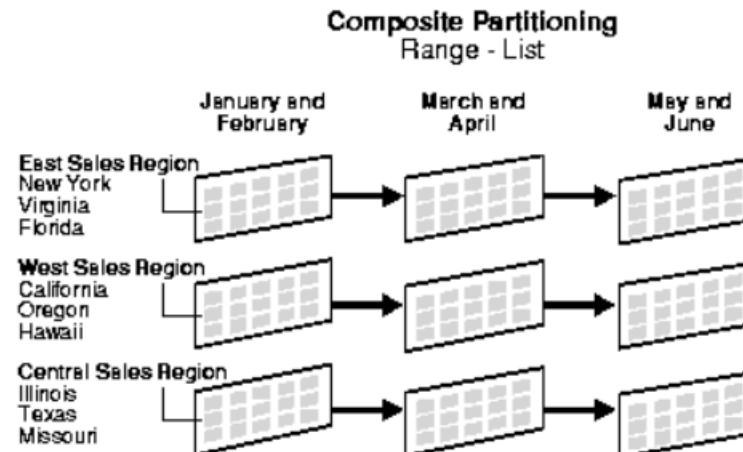
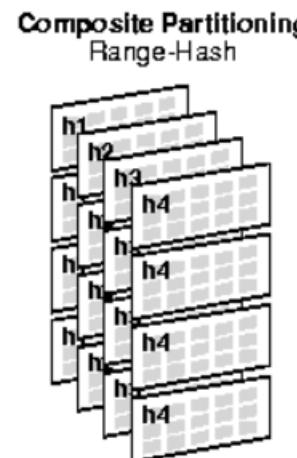
```
CREATE TABLE k1 (
    id INT NOT NULL PRIMARY KEY,
    name VARCHAR(20)
)
PARTITION BY KEY()
PARTITIONS 2;
```

If there is no primary key but there is a unique key, then the unique key is used for the partitioning key:

```
CREATE TABLE k1 (
    id INT NOT NULL,
    name VARCHAR(20),
    UNIQUE KEY (id)
)
PARTITION BY KEY()
PARTITIONS 2;
```

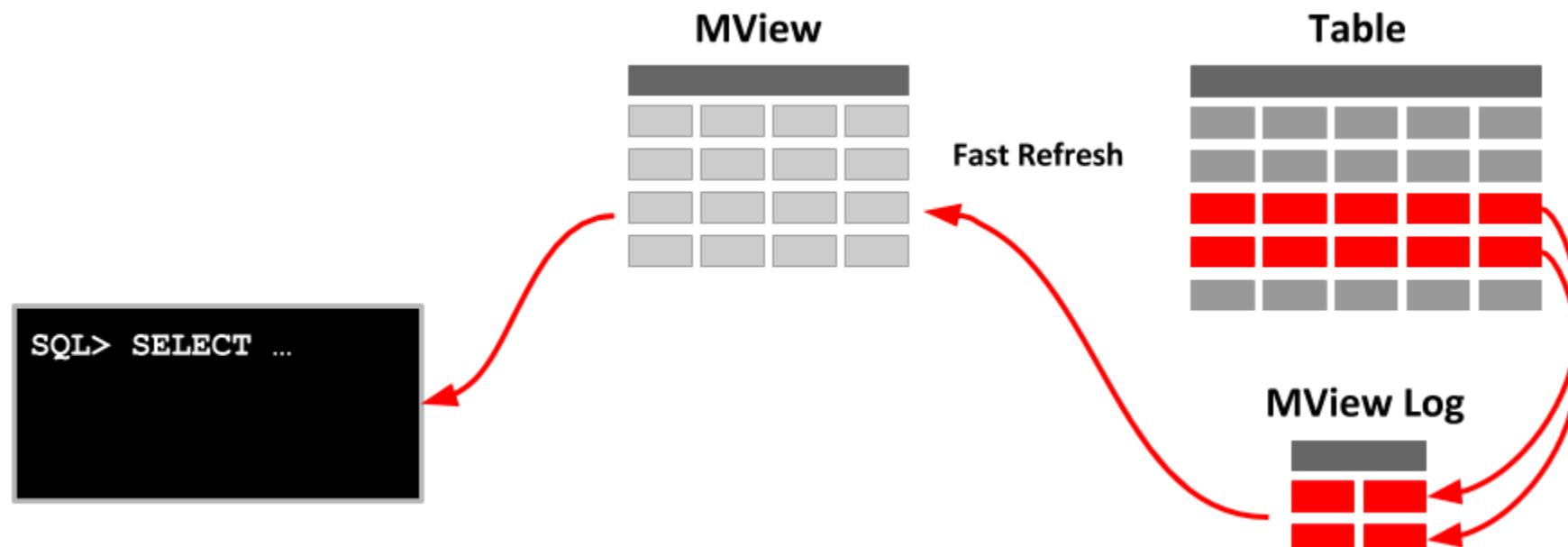
Composite Partitioning

- Composite partitioning combines range and hash partitioning.
- First distribute data into partitions according to boundaries established by the partition ranges.
- Then use a hashing algorithm to further divide the data into sub-partitions within each range partition.

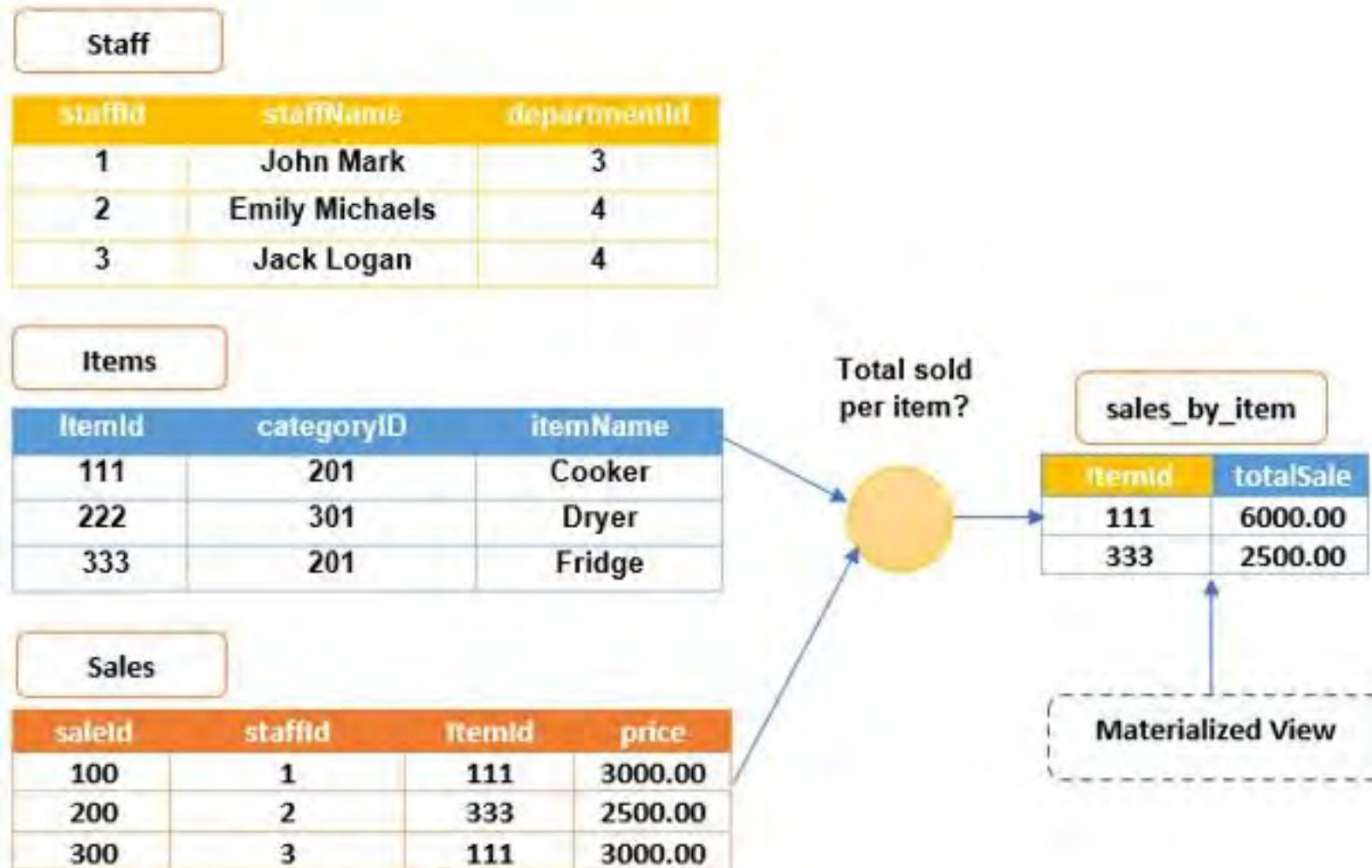


Materialized Views

- A materialized view, or snapshot as they were previously known, is a table segment whose contents are periodically refreshed based on a query, either against a local or remote table.
- To replicate data to non-master sites in a replication environment and to cache expensive queries in a data warehouse environment



Materialized Views- Example



Materialized Views- Basic Syntax

```
-- Normal
CREATE MATERIALIZED VIEW view-name
BUILD [IMMEDIATE | DEFERRED]
REFRESH [FAST | COMPLETE | FORCE ]
ON [COMMIT | DEMAND ]
[[ENABLE | DISABLE] QUERY REWRITE]
AS
SELECT ...;
```

```
-- Pre-Built
CREATE MATERIALIZED VIEW view-name
ON PREBUILT TABLE
REFRESH [FAST | COMPLETE | FORCE ]
ON [COMMIT | DEMAND ]
[[ENABLE | DISABLE] QUERY REWRITE]
AS
SELECT ...;
```

The **BUILD** clause options are shown below.

- **IMMEDIATE** : The materialized view is populated immediately.
- **DEFERRED** : The materialized view is populated on the first requested refresh.

Materialized Views- Basic Syntax

The following refresh types are available.

- FAST : A fast refresh is attempted. If materialized view logs are not present against the source tables in advance, the creation fails.
- COMPLETE : The table segment supporting the materialized view is truncated and repopulated completely using the associated query.
- FORCE : A fast refresh is attempted. If one is not possible a complete refresh is performed.

A refresh can be triggered in one of two ways.

- ON COMMIT : The refresh is triggered by a committed data change in one of the dependent tables.
- ON DEMAND : The refresh is initiated by a manual request or a scheduled task.

DDL Commands For Materialized Views



CREATE MATERIALIZED VIEW- Syntax

```
CREATE [ OR REPLACE ] [ SECURE ] MATERIALIZED VIEW [ IF NOT EXISTS ] <name>
[ COPY GRANTS ]
( <column_list> )
[ <col1> [ WITH ] MASKING POLICY <policy_name> [ USING ( <col1> , <cond_col1> , ... ) ]
      [ WITH ] TAG ( <tag_name> = '<tag_value>' [ , <tag_name> = '<tag_value>' , ... ] ) ]
[ , <col2> [ ... ] ]
[ [ WITH ] ROW ACCESS POLICY <policy_name> ON ( <col_name> [ , <col_name> ... ] ) ]
[ [ WITH ] TAG ( <tag_name> = '<tag_value>' [ , <tag_name> = '<tag_value>' , ... ] ) ]
[ COMMENT = '<string_literal>' ]
[ CLUSTER BY ( <expr1> [ , <expr2> ... ] ) ]
AS <select_statement>
```

DDL Commands For Materialized Views



ALTER MATERIALIZED VIEW

```
ALTER MATERIALIZED VIEW <name>
```

```
[  
  RENAME TO <new_name>  
  CLUSTER BY | <expr1> [, <expr2> ... ] |  
  DROP CLUSTERING KEY  
  SUSPEND RECLUSTER  
  RESUME RECLUSTER  
  SUSPEND  
  RESUME  
  SET [  
    [ SECURE ]  
    [ COMMENT = '<comment>' ]  
  ]  
  UNSET [  
    SECURE  
    COMMENT  
  ]  
]
```

DROP MATERIALIZED VIEW

```
DROP MATERIALIZED VIEW [ IF EXISTS ] <view_name>
```

DDL Commands For Materialized Views



SHOW MATERIALIZED VIEW

```
SHOW MATERIALIZED VIEWS [ LIKE '<pattern>' ]
[ IN
  [
    ACCOUNT
    |
    DATABASE
    DATABASE <database_name>
    |
    SCHEMA
    SCHEMA <schema_name>
    <schema_name>
  ]
]
```

Bitmap Indexes

- Bitmap indexes are widely used in data warehousing applications, which have large amounts of data and ad hoc queries but a low level of concurrent transactions.
- For such applications, bitmap indexing provides:
 - Reduced response time for large classes of ad hoc queries
 - A substantial reduction of space usage compared to other indexing techniques.
- The purpose of an index is to provide pointers to the rows in a table that contain a given key value.
- In a regular index, this is achieved by storing a list of *rowids* for each key corresponding to the rows with that key value. In a bitmap index, a bitmap for each key value is used instead of a list of *rowids*

Bitmap Indexes- Example

CUSTOMER #	MARITAL_ STATUS	REGION	GENDER	INCOME_ LEVEL
101	single	east	male	bracket_1
102	married	central	female	bracket_4
103	married	west	female	bracket_2
104	divorced	west	male	bracket_4
105	single	central	female	bracket_2
106	married	central	female	bracket_3



bitmap index for the REGION column

REGION='east'	REGION='central'	REGION='west'
1	0	0
0	1	0
0	0	1
0	0	1
0	1	0
0	1	0

Dimensions

- Before you can create a dimension object, the dimension tables must exist in the database possibly containing the dimension data
- You create a dimension using the CREATE DIMENSION statement
- For example, you can declare a dimension *products_dim*, which contains levels product, subcategory, and category:

```
CREATE DIMENSION products_dim
LEVEL product IS (products.prod_id)
LEVEL subcategory IS (products.prod_subcategory)
LEVEL category IS (products.prod_category) ...
```

- Next step is to specify the hierarchy:
HIERARCHY prod_rollup
(product CHILD OF
subcategory CHILD OF
category)

SQL

- Rollup
- Cube
- Window Queries
- Top N Queries
- Typically, a single OLAP operation can lead to several closely related SQL queries with aggregation and grouping
- Cross-tabulation is an example!

Cube Operator

Locid	City	State	Country
1	Madison	WI	USA
2	Fresno	CA	USA
5	Chennai	TN	India

Pid	Pname	category	Price
11	Lee Jeans	Apparel	25
12	Zord	Toys	18
13	Biro Pen	Stationery	2

Timeid	Date	Month	Year	Holiday
1	10/11/05	Nov	1995	N
2	11/11/05	Nov	1996	N
3	12/11/05	Nov	1997	N

pid	timeid	Locid	sales
11	1	1	25
11	2	1	8
11	3	1	15
12	1	1	30
12	2	1	20
12	3	1	50
13	1	1	8
13	2	1	10
13	3	1	10
11	1	2	35
11	2	2	22
11	3	2	10
12	1	2	26
12	2	2	45
12	3	2	20
13	1	2	20
13	2	2	40
13	3	2	5

Cube Operator

```
Select T.year, L.state, SUM(sales)
from Sales S, Times T, Locations L
Where S.timeid=T.timeid & S.locid=L.locid
Group By T.year, L.state
```

```
Select T.year, SUM(sales)
from Sales S, Times T
Where S.timeid=T.timeid
Group By T.year
```

```
Select L.state, SUM(sales)
from Sales S, Locations L
Where S.locid=L.locid
Group By L.state
```

	WI	CA	Total
1995	63	81	144
1996	38	107	145
1997	75	35	110
Total	176	223	399

Select SUM(sales)

from Sales S, Locations L

Where S.locid=L.locid

OR

Select SUM(sales)

from Sales S, Time T

Where S.timeid=T.timeid

How many such SQL queries to build cross-tab?

Cube Operator

- Cross-tab can be thought of as a roll-up on the entire dataset, on location, on time, and on both location and time dimensions together
- Each roll-up corresponds to a single SQL query with grouping
- Given a FACT with k associated dimensions, we will have a total of 2^k such SQL queries
- GROUP BY construct is extended to provide better support for roll-up and cross-tabulation queries
- Collection of GROUP BY statement is equivalent to GROUP BY with CUBE keyword, with one GROUP BY statement for each subset of the k dimensions

Cube Operator

```
Select T.year, L.state, SUM(sales)
from Sales S, Times T, Locations L
Where S.timeid=T.timeid & S.locid=L.locid
Group By CUBE (T.year, L.state)
```

	WI	CA	Total
1995	63	81	144
1996	38	107	145
1997	75	35	110
Total	176	223	399

T.Year	L.State	SUM(sales)
1995	WI	63
1995	CA	81
1995	All	144
1996	WI	38
1996	CA	107
1996	All	145
1997	WI	75
1997	CA	35
1997	All	110
All	WI	176
All	CA	223
All	All	399

Rollup Operator

```
Select T.year, L.state, SUM(sales)
from Sales S, Times T, Locations L
Where S.timeid=T.timeid & S.locid=L.locid
Group By ROLLUP (T.year, L.state)
```

	WI	CA	Total
1995	63	81	144
1996	38	107	145
1997	75	35	110
Total	176	223	399

Find out what the following SQL will generate?

```
Select T.year, L.state, SUM(sales)
from Sales S, Times T, Locations L
Where S.timeid=T.timeid & S.locid=L.locid
Group By ROLLUP (L.state, T.year)
```

T.Year	L.State	SUM(sales)
1995	WI	63
1995	CA	81
1995	All	144
1996	WI	38
1996	CA	107
1996	All	145
1997	WI	75
1997	CA	35
1997	All	110
All	All	399

Window Queries in SQL

- Time dimension is very important in decision support
- Queries involving trend analysis have been difficult to express in SQL
- A fundamental extension called a “query window” is introduced.
- Window functions applies aggregate and ranking functions over a particular window (set of rows).
- OVER clause is used with window functions to define that window.
- OVER clause does two things :
 - Partitions rows into form set of rows. (PARTITION BY clause is used)
 - Orders rows within those partitions into a particular order. (ORDER BY clause is used)

Window Queries- Syntax

```
SELECT column_name1,  
       window_function(column_name2)  
    OVER([PARTITION BY column_name1] [ORDER BY column_name3]) AS  
        new_column  
   FROM table_name;
```

window_function= any aggregate or ranking function

column_name1= column to be selected

column_name2= column on which window function is to be applied

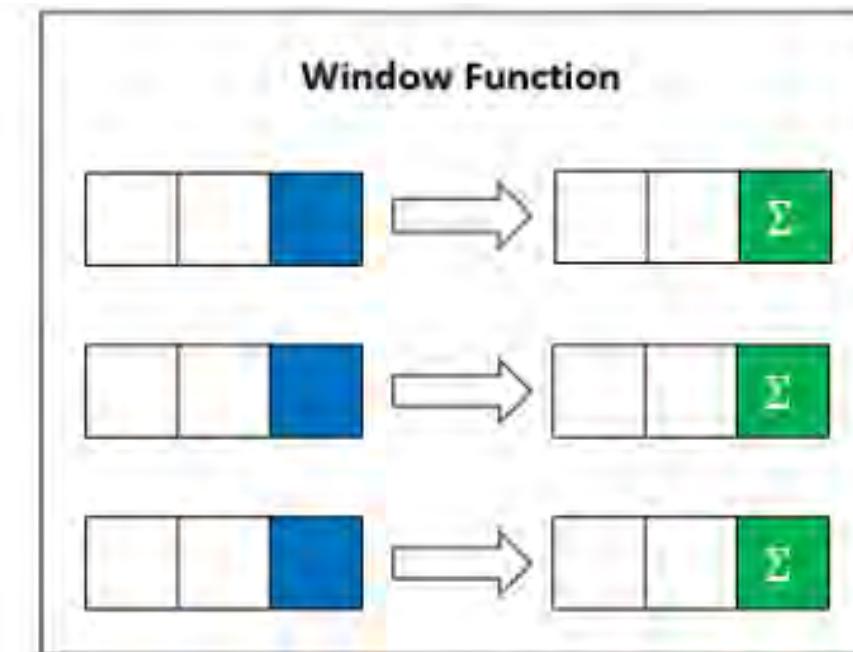
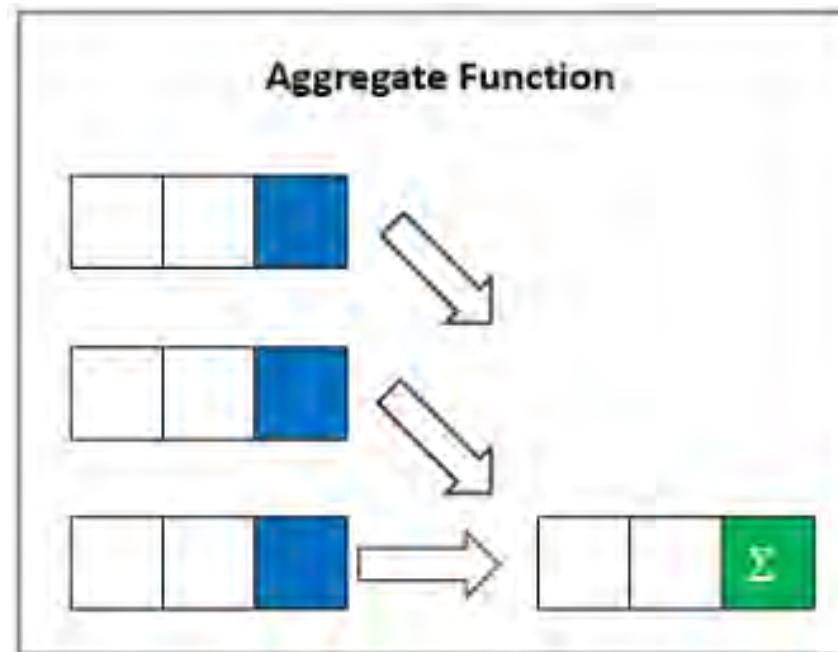
column_name3= column on whose basis partition of rows is to be done

new_column= Name of new column

table_name= Name of table

Window Queries- Syntax

- Difference between aggregate functions and window functions:



Window Queries- Example

We can associate the avg. sales over the past 3 days with every sales tuple (daily granularity)

This gives a 3-day moving avg. of sales

```
SELECT L.state, T.month, AVG(S.sales) OVER W AS movavg
FROM Sales S, Times T, Locations L
WHERE S.timeid=T.timeid AND S.locid=L.locid
WINDOW W AS (PARTITION BY L.state
              ORDER BY T.month
              RANGE BETWEEN INTERVAL '1' MONTH PRECEDING
              AND INTERVAL '1' MONTH FOLLOWING)
```

- FROM & WHERE clauses proceed as usual to generate an intermediate table, TEMP.
- WINDOWS are created over TEMP
- 3 steps in defining a window
 - Define partitions of the table (Partitions are similar to groups created by GROUP BY)
 - Specify the ordering of rows within a partition
 - Frame WINDOW: establish the boundaries of the window associated with each row in terms of ordering of rows within partitions

Window Queries- Example

```
SELECT L.state, T.month, AVG(S.sales) OVER W AS movavg  
FROM Sales S, Times T, Locations L  
WHERE S.timeid=T.timeid AND S.locid=L.locid  
WINDOW W AS (PARTITION BY L.state  
              ORDER BY T.month  
              RANGE BETWEEN INTERVAL '1' MONTH PRECEDING  
                AND INTERVAL '1' MONTH FOLLOWING)
```

- Define partitions of the table (Partitions are similar to groups created by GROUP BY)
- Specify the ordering of rows within a partition
- Frame WINDOW: establish the boundaries of the window associated with each row in terms of ordering of rows within partitions
- Window for each row includes the row itself, plus all rows whose month values are within a month before or after.
- A row whose month value is June 2006 has a window containing all rows with month = May, June, or July 2006

Window Queries- Example

```
SELECT L.state, T.month, AVG(S.sales) OVER W AS movavg  
FROM Sales S, Times T, Locations L  
WHERE S.timeid=T.timeid AND S.locid=L.locid  
WINDOW W AS (PARTITION BY L.state  
              ORDER BY T.month  
              RANGE BETWEEN INTERVAL '1' MONTH PRECEDING  
                AND INTERVAL '1' MONTH FOLLOWING)
```

- Answer rows to each row is constructed first by identifying its WINDOW
- Then, for each answer column defined using a window agg. fn, we compute the agg. Using the rows in the WINDOW
- Each row of TEMP is a row of sales, tagged with extra details about time & location dimensions
- One partition for each state and every row of temp belongs to exactly one partition.

Top N Queries

- If you want to find the 10 (or so) cheapest cars, it would be nice if the DB could avoid computing the costs of all cars before sorting to determine the 10 cheapest.
 - **Idea:** Guess at a cost c such that the 10 cheapest all cost less than c , and that not too many more cost less. Then add the selection $\text{cost} < c$ and evaluate the query.
 - If the guess is right, great, we avoid computation for cars that cost more than c .
 - If the guess is wrong, need to reset the selection and recompute the original query.

Top N Queries

```
SELECT P.pid, P.pname, S.sales  
FROM Sales S, Products P  
WHERE S.pid=P.pid AND S.locid=1 AND S.timeid=3  
      AND S.sales > c  
ORDER BY S.sales DESC
```

- OPTIMIZE FOR construct is not in SQL:92 & not even in SQL:1999!
- Supported by IBM's DB2 & Oracle 9i has similar constructs
- Compute sales only for those products that are likely to be in TOP 10

Top N Queries

```
SELECT P.pid, P.pname, S.sales  
FROM Sales S, Products P  
WHERE S.pid=P.pid AND S.locid=1 AND S.timeid=3  
      AND S.sales > c  
ORDER BY S.sales DESC
```

- Cut-off value c is chosen by optimizer using the histogram on the sales column of the sales relation.
- Much faster approach
- Issues:
 - How to choose c ?
 - What if we get more than 10 products?
 - What if we get less than 10 products?

Difference between Database System and Data Warehouse



Database System	Data Warehouse
It supports operational processes.	It supports analysis and performance reporting.
Capture and maintain the data.	Explore the data.
Current data.	Multiple years of history.
Data is balanced within the scope of this one system.	Data must be integrated and balanced from multiple system.
Data is updated when transaction occurs.	Data is updated on scheduled processes.
Data verification occurs when entry is done.	Data verification occurs after the fact.
100 MB to GB.	100 GB to TB.
ER based.	Star/Snowflake.
Application oriented.	Subject oriented.
Primitive and highly detailed.	Summarized and consolidated.
Flat relational.	Multidimensional.

SQL Extensions

LATEST ON

LATEST ON is a clause introduced to help find the latest entry by timestamp for a given key or combination of keys as part of a SELECT statement.

LATEST ON customer ID and currency

```
SELECT * FROM balances  
WHERE balance > 800  
LATEST ON ts PARTITION BY customer_id, currency;
```

SQL Extensions

SAMPLE BY

- SAMPLE BY is used for time-based aggregations with an efficient syntax.
- The short query below will return the simple average balance from a list of accounts by one month buckets

SAMPLE BY one month buckets

```
SELECT avg(balance) FROM accounts SAMPLE BY 1M
```

SQL Extensions

TIMESTAMP SEARCH

Timestamp search can be performed with regular operators, e.g >, <= etc

Results in a given year

```
SELECT * FROM scores WHERE ts IN '2018';
```

IMP Note to Self





Thank you



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Privacy

Pravin Y Pawar

Adapted from “Reliable Machine Learning”
By

Privacy

- Privacy is a notion that has proven notoriously difficult to define in scholarship
- The rise of big data has proven a dramatic example of Privacy.
- Experts used to think that de-identified data was appropriate and safe to release
 - Such data would have removed what was thought to be identifying information, such as name or address, that could easily be matched to a person
 - However, information about other factors associated with, but seemingly not directly related to, a specific person's identity was still released, such as birthday, race, or zip code.
- With the advent of big data, many datasets were compiled, often about overlapping groups of people,
 - became possible to use different datasets together to identify people from de-identified information
 - For example, for most Americans, it turned out to be possible to identify them in a de-identified dataset merely from knowing their birthday, zip code, and gender.

Privacy goals in ML

- An ML process involves many assets which may need to be protected such as
 - The identity of the data contributors
 - the raw dataset collected by them
 - the feature extracted from datasets
 - the model itself
 - its input (or part of it) whenever the model is queried

Privacy goals in ML(2)

- Identity Privacy
 - Protecting the data contributor's identity is necessary in many data collection processes,
 - especially when the data collected may reveal political beliefs, sexual preferences, health conditions etc.
 - Achieving identity privacy can be seen as ensuring anonymity of the data contributors
 - both from the model creators and the model owners, who are highly involved in the data collection stage of an ML process,
 - but also from adversaries which in many cases are acting as model consumers
- Raw Dataset Privacy
 - Often, the raw dataset Z contains sensitive information, such as the X-ray images from the cancer diagnosis example
 - Privacy violations could also introduce further new attack vectors
 - For instance, consider a scenario where an ML model can calculate energy efficient plans
 - previously it has been trained on a dataset which contains the energy consumption and location of houses
 - Compromising this dataset can expose the time intervals in which the residents of a specific house are absent
 - The latter can then be used as input in defining strategies for robberies
 - Often, compromising the privacy of the raw dataset could also result in compromising identity privacy,
 - since data in Z can be used to identify a specific data contributor
 - A common malpractice which eases the task of compromising Z is storing the dataset as plaintext
 - instead of using state of the art encryption

Privacy goals in ML(3)

- Feature Datasets Privacy
 - The privacy of the feature datasets is as important as the privacy of the raw dataset
 - Both the training set X_1 and the validation set X_2 are extracted from Z
 - in case of them being compromised, information about Z could also be recovered
 - Unfortunately, the privacy of the feature datasets is threatened in many ML models, which by default store the set X_1 inside the model
- Model Privacy
 - Privacy of the model $\mathcal{M}\theta$ refers to the secrecy of the produced model $\mathcal{M}\theta$ and its parameter θ
 - keeping the model secret is a common goal desired by the model owners
 - However, exposing part of the model's functionality to the model consumers is inevitable
 - since observing the responses to queries may reveal information about the internal structure of the model
- Input Privacy
 - Often the input (or part of it) to the model may be sensitive and needs to be protected
 - only the output of the model is revealed to the model consumers
 - Consider the scenario of an ML model that takes as input medical records to predict if a certain disease can be inherited
 - likely that the medical record contributors do not desire to disclose their medical records to each other
 - In many cases, achieving input privacy requires not only keeping the input x secret,
 - but also limiting the information about x disclosed through the response $y = \mathcal{M}\theta$

Two key ideas

- In terms of
 - how they relate to a notion of privacy
 - how they relate to taking an existing dataset with personal information and turning it into a dataset that can be used or released without compromising individual identity
- Broad Ideas
 - k-anonymity
 - Differential privacy

Threat Models and Attacks

- Adversary models and Type of attacks that target the assets of ML processes
- Two different adversary models
 - White box adversaries
 - Black box adversaries
- White box adversaries
 - who know the structure of the model $\mathcal{M}\theta$, its parameter θ ,
 - while they may also know part of the raw dataset Z
 - finally, they can interact with the model (i.e., the adversary is a model consumer in this case)
- Black box adversaries
 - who have no knowledge about the model and its parameters
 - but they can query the model and observe its response

Threat Models and Attacks(2)

- Membership Inference Attacks
 - Can be performed by a black box adversary
 - Adversary wants to determine if a given data point z was part of the raw dataset Z , or if a particular data point x was part of the training set X_1
 - Usually exploit that ML models often behave differently on the data used for their training versus unseen data
 - Can compromise the raw dataset privacy or the feature datasets privacy
 - For instance,
 - this attack could be used by an adversary to learn
 - whether a specific individual's record was used to train an ML model
 - which determines the presence of a certain disease
- De-anonymisation Attacks
 - Aims to identify an individual who has contributed his data into a dataset
 - Many model owners publish the raw dataset Z or the feature datasets X_1 and X_2 ,
 - which may allow an adversary to compromise the identity privacy of the data contributors
 - even though the dataset has been first anonymized
 - Requires white box access to the dataset

Threat Models and Attacks(3)

- Reconstruction Attacks
 - Goal of this attack is to construct the raw dataset Z
 - by reverse engineering the feature training dataset X_1 or the validation dataset X_2
 - Can compromise the raw dataset privacy
 - Requires white box access to a model that hard-codes the feature datasets inside it
- Model Extraction Attacks
 - Goal of this attack is to construct an approximation $\hat{M}\theta$ of the model $M\theta$
 - The more accurate the approximation is, the more successful the attack is
 - Adversary has black box capabilities and trains his model $\hat{M}\theta$ by collecting pairs of queries and their responses received by the model $M\theta$
 - Violates the model privacy
- Model Inversion Attacks
 - Aim at inferring properties about the training dataset X_1 , even when it is not explicitly stored in the model $M\theta$
 - Can even be performed by a limited black box attacker who can interact with the model by querying it and collecting its responses
 - Adversary may first perform a model extraction attack and then utilise the model $\hat{M}\theta$ to accomplish a model inversion attack

k-anonymity

- The idea behind k-anonymity is that in a given dataset,
 - for any given combination of categories of interest,
 - there should be at least k individuals (externally specified) who fall into any given bucket
- For example, could apply k-anonymity to a dataset listing individuals in a town
 - by requiring that data be bucketed such that for any zip code / birth information / gender category there were at least 10 individuals
- Many ways to accomplish this
 - report birthdays at the month or year level
 - report only the first four digits of a zip code rather than all five
- The appropriate size for k will depend on the particular domain,
 - which could have different implications regarding sensitivity of the data,
 - possibility to build useful datasets with large k, and possibility of reidentification given other known potentially linkable datasets

Differential privacy

- Mathematical method adds noise to data such that probabilistic guarantees can be made regarding the possibility (or more importantly, lack of possibility) to make inferences about a specific individual when given access to that noisified data
- Idea behind differential privacy is that it would be a privacy problem for one individual's inclusion, or non-inclusion, in a dataset to influence operations, such as the calculation of averages or other statistics, so that the dataset's information could be inferred based on aggregate reporting
- For example,
 - if the mean age of a class is reported and the size of the group, with and without an individual, possible to know that individual's age
 - If differential privacy is applied, it would, in probability, not be possible to infer that individual's age at a pre-specified level of precision and given a pre-specified query budget

Methods to Preserve Privacy

- k-anonymity and differential privacy refer to very specific notions of privacy and computational measures
 - Indeed, privacy is itself a highly technical and specialized field, perhaps best left to experts in terms of implementation
- A variety of accessible privacy-enhancing measures are transparent, straightforward to implement, and meaningful
 - Should include them in workflow and will likely need to customize them
- Likely to already be familiar concepts to systems administrators and those who deal with compliance issues
 - sometimes are painfully unfamiliar to data scientists and ML engineers
- Two aspects
 - Technical measures
 - Institutional measures

Technical measures

- Access controls
 - A key way to preserve privacy and reduce threats is to implement robust access controls
 - Any data about people in a database should be treated as a “need to know” resource,
 - with ML engineers requesting access for specific purposes
 - rather than being able to freely access or browse data
 - Likewise, access should be revisited periodically to make sure
 - staff members do not retain access to data for which they no longer have a valid active reason for access
- Access logging
 - Keep track of who is accessing specific forms of data and when
 - Makes it possible to understand
 - data use patterns,
 - see when someone might be inappropriately accessing data,
 - preserve evidence in case allegations of inappropriate use are made later
 - An analysis of such logs may also indicate ways
 - in which data storage schema could be refactored to reduce the extent of data that different use patterns can access
 - For example, if an ML modeling calls for access to a sensitive table of data merely to access one column,
 - consider splitting off that column of data rather than granting access to a full table of additional but unnecessary pieces of information

Technical measures(2)

- Data minimization
 - The collection of data should be minimized, and likewise the use of data should be minimized
 - Data should not be collected merely because it “could be useful” in the future
 - Data should be logged only when there is an immediate use case for this data
- Data separation
 - The data needed for legitimate business uses should be separated from sensitive data(such as names and addresses) that is unlikely to be relevant to creating an ML model
 - For example, to predict users’ clicks, there doesn’t seem to be a justification for knowing a user’s name or address
 - no reason for that information to be stored with information that might be useful for that particular prediction task, such as past browsing history or demographic information.
 - Studying data access logs can help to identify ways in which data storage can be refactored to minimize exposure of data to ML applications

Institutional measures

- Ethics training
 - Everyone needs ethics training when they enter for designing ML products
 - ML engineers should be given — but usually are not — a thorough review
 - not only of general ethics training (such as the possibility of bias in data)
 - but also domain-specific training when building algorithms for a specific use case
- Data access guidelines
 - Makes sense to have explicit rules that are readily available regarding
 - what constitutes appropriate access to data and use of that data and what use cases are expressly prohibited
 - A lack of clear and explicit ethics rules
 - can lead to an institutional culture without accountability
 - Should have clear data access and appropriate data use guidelines in place in a location that is readily apparent and accessible

Institutional measures(2)

- Privacy by design
- Set of design principles that can apply to digital product, including ML pipelines and ML-driven products
- The notion is that privacy is something that shouldn't be tacked on to the end of a process that already exists
- Rather, privacy should be intrinsic to design considerations from the start
 - should be a question and concern addressed at all working stages
- Privacy by design can provide a flexible but holistic way to ensure privacy in all elements of ML development



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Privacy-preserving Techniques

Pravin Y Pawar

Adapted from PRIVACY-PRESERVING MACHINE LEARNING
by Zaruhi Aslanyan, Panagiotis Vasilikos

Privacy-preserving techniques

- Privacy-preserving techniques that can be used to strengthen ML algorithms to protect individuals' data
 - Anonymisation
 - Differential Privacy
 - Homomorphic Encryption
 - Multi-party Computation
 - Federated Learning



Anonymization

- Data anonymization aims at protecting the privacy of an individual in a given dataset
- Before releasing a dataset,
 - First step is to remove all direct identifiers, such as name, address and phone numbers
 - in most of the cases this is not enough
 - Only removing the direct identifiers will still allow an attacker to re-identify an individual in the dataset by linking the data with other additional information that he/she has
- In order to avoid such cases, additional anonymization steps should be enforced
 - Several methods, such as k -anonymity and l -diversity, have been studied
 - that make it harder for an adversary to learn anything about an individual from an anonymized dataset
- For a great many cases l -diversity together with k -anonymity provides a robust privacy model
 - k -anonymity is an anonymization method that ensures the information about an individual in the published dataset cannot be distinguished from at least $k-1$ other individuals in the same dataset
 - On the other hand, a dataset is called l -diverse if every equivalence class has at least $l \geq 2$ different values for the sensitive attributes
- Though challenging, anonymization needs to be applied aiming at providing data privacy while preserving utility.

Differential Privacy

- Differential privacy is a privacy method that helps reveal useful information about a dataset without revealing any private information about an individual in the dataset
 -
- The method guarantees that even if an attacker knows all records in a dataset,
 - s/he will not be able to link a specific record to an individual on the basis of the output of a differentially-private method
- Outcome of the analysis is not(significantly) dependent on an individual's record being in the dataset
 - Hence, the privacy risk is essentially the same with or without the individual's participation in the dataset
- Differential privacy is achieved by adding random noise to the result,
 - which can be done through various differentially-private mechanisms,
 - such as the Laplace mechanism, the exponential mechanism and the randomized response mechanism

Homomorphic Encryption

- Cryptographic method that allows computing on encrypted data,
 - decrypted output is identical to the output of the computation on the original unencrypted input
- The method is typically used as follows:
 1. the owner of the data encrypts them by means of a homomorphic function and shares the result with a third party in charge of performing a given computation
 2. the third party performs the computation on the encrypted data and returns the result, which is encrypted because the input data are encrypted
 3. the owner of the data decrypts the result, thereby obtaining the result of the computation on the original plain-text data
- During this process the third party does not have access to the unencrypted input or output.
- Has a number of limitations
 - various encryption schemes that allow performing different computations on encrypted data,
 - they are generally limited to working with addition or multiplication over integers
- For example, an additively encryption scheme allows to take two encrypted messages, m_1 and m_2 , and produce the encrypted message $m_1 + m_2$.
 - Moreover, for security purposes, some noise is typically added to the input data during the encryption process
 - noise accumulates with the operations performed on encrypted data
 - noise growth is particularly severe under multiplication
- In order to get the expected result when decrypting, the noise must be kept below a certain threshold
 - This threshold affects the number of computations that can be performed on encrypted data

Multi-party Computation (MPC)

- Technology that allows multiple parties to compute a function on their private inputs without revealing them
 - parties are independent and do not trust each other
 - main idea is to allow to perform a computation on the private data while keeping the data secret
 - guarantees that all participants learn nothing more than what they can learn from the output and their own input
- Both MPC and homomorphic encryption are powerful privacy techniques,
 - however they usually have high communication and computation costs

Multi-party Computation (MPC) 2

- Garbled circuits
 - a widely-used cryptographic protocol for two-party secure computation on Boolean functions (circuits)
 - The steps of the protocol are as follows:
 - the first party, Alice, encrypts (or garbles) the function (circuit) and sends it to the second party, Bob, together with her encrypted input
 - with the help of Alice, Bob encrypts his own input using oblivious transfer, i.e., Alice and Bob transfer some information such that the sender remains oblivious what information has been transferred
 - Bob evaluates the function using both encrypted inputs and gets the encrypted output
 - finally, the two parties communicate to learn the output
- Secret sharing
 - method common to many MPC approaches
 - A (t, n) -secret sharing method divides the secret s into n shares and allocates a share to a participant
 - Secret s can be reconstructed by combining together t shares, while no information about s will be revealed when any $t-1$ of the shares are combined
 - Secret is divided in such a way that any group of at least t participants can reconstruct the secret, while no group of less than t participants can.

Federated Learning

- Federated learning allows ML processes to be decentralised limiting the information exposed from the contributor's datasets
 - controlling the risk of compromising the dataset's and identity privacy
- General idea of federated learning
 - A central ML model $\mathcal{M}\theta$ owned by some central authority (e.g., a company) can be further trained on new private datasets from data contributors
 - by having each contributor performing the training locally with its own dataset
 - then updating the central model $\mathcal{M}\theta$ (i.e., update the model's parameter θ)
- Works as follows
 - (1) the central model $\mathcal{M}\theta$ is distributed among a number of n participants (i.e., the data contributors)
 - (2) each participant updates locally the model $\mathcal{M}\theta$ by training it on its own local dataset Zl , producing a new local parameter θl
 - (3) each participant sends its update θl to the central authority
 - (4) the central authority aggregates the local parameters of each participant, producing a new parameter θ' which is used to update the central model
 - (5) This process can be further continued until the central model has been trained enough
- Can be seen as a flexible approach for achieving privacy of raw datasets, feature datasets and identity privacy,
 - since it lifts the requirement of collecting vast amounts of information from the data contributors and then train on them in a centralised manner
- Communication in federated learning networks could create performance overhead, which is not present in centralised ML approaches,
 - put in conjunction with the availability of devices in the network, scalability also becomes a concern



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Monitoring and Observability

Pravin Y Pawar

Adapted from [Monitoring and Observability](#)
by Cindy Sridharan

Only half joking – Monitoring to Observability

- Why call it monitoring? That's not sexy enough anymore.
- Observability, because rebranding Ops as DevOps wasn't bad enough, now they're devopsifying monitoring too
- Is that supposed to be like the second coming of DevOps? Or was it the Second Way? I can't remember. It all felt so cultish anyway.
- **What is the difference between “monitoring” and “observability”, if any?**
 - Or is the latter just the latest buzzword on the block, to be flogged and shoved down throats until it has been milked for all its worth

Monitoring

- “Monitoring” traditionally was a preserve of Operations engineers
 - term often invokes not very pleasant memories in minds of many who’ve been doing it for long enough
- In the eyes of many, “monitoring” harks back to many dysfunctional aspects of the old school way of operating software,
 - term this day causes some people to think of simple up/down checks
- True that a decade ago, up/down checks might’ve been all a “monitoring” tool would have been capable of
 - in the recent years “monitoring” tools have evolved greatly
 - to the point where many, many, many people no longer think of monitoring as just external pings
- While they might still call it “monitoring”,
 - the methods and tools they use are more powerful and streamlined

Whitebox monitoring

- In general, tend to measure at three levels: network, machine, and application
 - Application metrics are usually the hardest, yet most important, of the three
 - Very specific to business, and they change as applications change
- Time series, logs and traces are all more in vogue than ever these days
 - are forms of “whitebox monitoring”, which refers to a category of monitoring based on the information derived from the internals of systems
- Whitebox monitoring isn’t really a revolutionary idea anymore
 - at least not since Etsy published the [seminal blog post](#) introducing statsd
- For all its flaws, statsd was a game changer,
 - most open source time series based systems as well as commercial monitoring solutions have supported statsd style metrics for years now
 - while not perfect, statsd style metrics collection was a huge improvement over the way one did “monitoring” previously

Monitoring is for symptom based Alerting

- Monitoring system should address two questions: what's broken, and why?
 - the “what's broken” indicates the symptom
 - the “why” indicates a (possibly intermediate) cause
 - “What” versus “why” is one of the most important distinctions in writing good monitoring with maximum signal and minimum noise.
- Blackbox monitoring
 - monitoring a system from the outside by treating it as a blackbox — is something is very good at answering the what is broken and alerting about a problem that's already occurring (and ideally end user impacting)
- Whitebox monitoring,
 - is fantastic for the signals that can be anticipated in advance and be on the lookout for
 - is for the known, hard failure modes of a system, the sort that lend themselves well toward exhibiting any deviation in a dashboardable manner, as it were.

“Monitorable” systems

One of the design goals while building systems should be to make it as monitorable as possible

- A good example of something that needs “monitoring”
 - a storage server running out of disk space
 - a proxy server running out of file descriptors
- Building “monitorable” systems requires being able to understand the failure domain of the critical components of the system proactively!
- The more mature a system, the better understood are its failure modes
 - Being able to entirely architect away a failure mode is the best thing one can do while building systems
 - The next best thing one can do is to be able to “monitor” impending failures and alert accordingly
- For monitoring to be effective, it becomes salient to be able to identify
 - a small set of hard failure modes of a system
 - or a core set of metrics that can indicate the health of the system accurately

Monitoring data needs to actionable

- In choosing what to monitor, keep the following guidelines to be considered:
 - The rules that catch real incidents most often should be as simple, predictable, and reliable as possible.
 - Data collection, aggregation, and alerting configuration that is rarely exercised (e.g., less than once a quarter for some SRE teams) should be up for removal.
 - Signals that are collected, but not exposed in any prebaked dashboard nor used by any alert, are candidates for removal.
- When not used to directly drive alerts, monitoring data should be optimized for providing a bird's eye view of the overall health of a system
 - In the event of a failure, monitoring data should immediately be able to
 - provide visibility into impact of the failure as well as the effect of any fix deployed
- The crucial thing to understand here is that monitoring doesn't guarantee that failure can be completely avoided
 - Monitoring provides a good approximation of the health of a system, but monitoring doesn't prevent failure entirely!
- Monitoring can furnish one with a panoramic view of systems' performance and behavior in the wild
 - however, does not make systems completely impregnable to failure, and that shouldn't be its goal either.

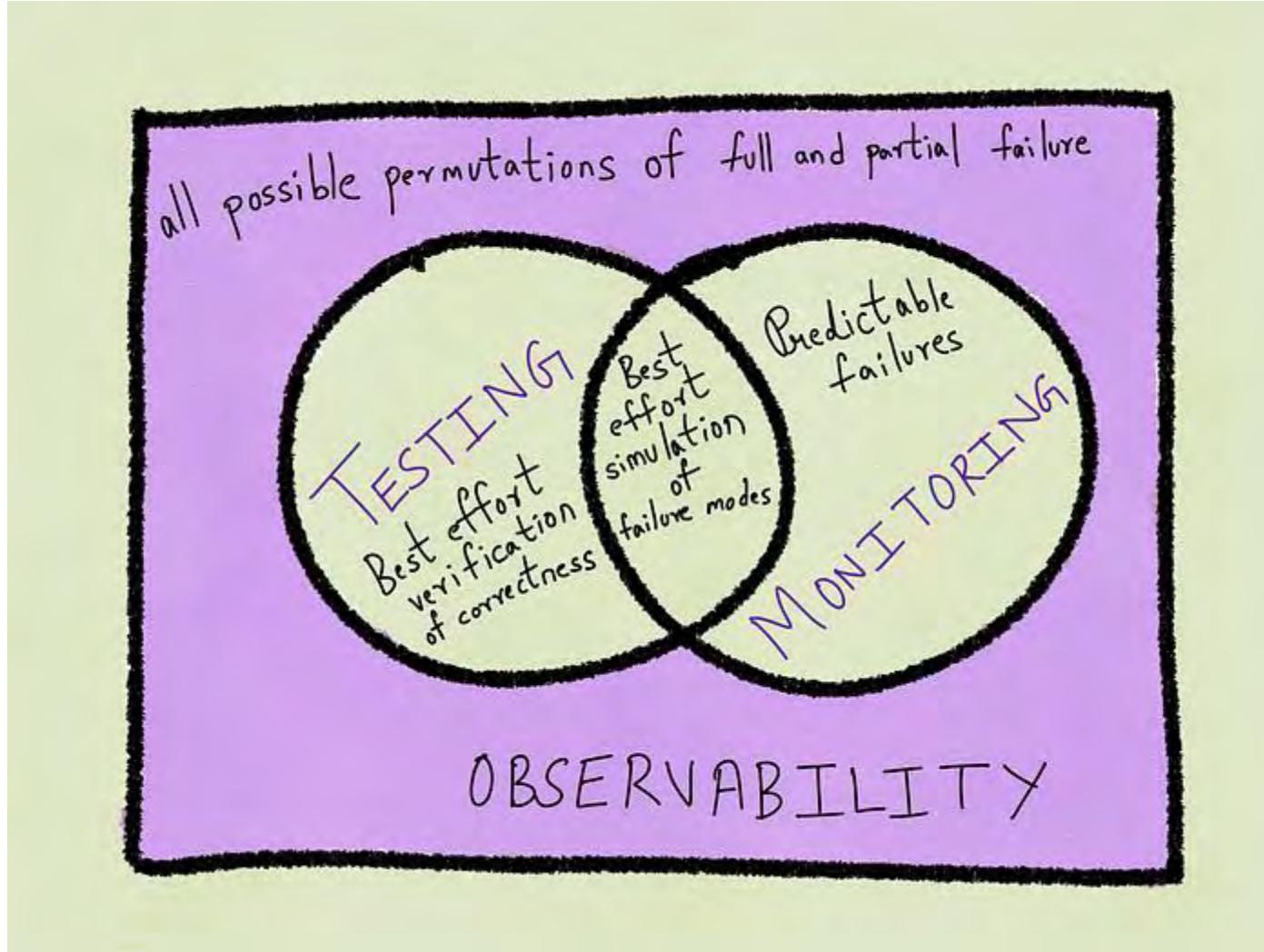
Monitoring vs Observability

- The goals of “monitoring” and “observability” are different!
 - “Observability” isn’t a substitute for “monitoring” nor does it obviate the need for “monitoring”; they are complementary.
- “Observability” might be a fancy new term on the horizon, but it really isn’t a novel idea
 - Events, tracing, exception tracking are all a derivative of logs
 - if one has been using any of these tools, one already has some form of “observability”
- In essence “observability” captures what “monitoring” doesn’t (and ideally, shouldn’t)!

Monitoring vs Observability(2)

- “Monitoring” is best suited to report the overall health of systems
 - Aiming to “monitor everything” can prove to be an anti-pattern
 - is best limited to key business and systems metrics derived from time-series based instrumentation, known failure modes as well as blackbox tests
- “Observability”, on the other hand, aims to provide highly granular insights into the behavior of systems along
 - with rich context,
 - perfect for debugging purposes
- Since it’s still not possible to predict every single failure mode a system could potentially run into or predict every possible way in which a system could misbehave,
 - it becomes important that we build systems that can be debugged armed with evidence and not conjecture

Monitoring vs Observability(3)



[source](#)

Observability at Twitter

- Twitter has since published [a two part blog post](#) on its observability stack
 - The posts are more about the architecture of the different components than the term itself
- These are the four pillars of the Observability Engineering team's charter:
 - Monitoring
 - Alerting/visualization
 - Distributed systems tracing infrastructure
 - Log aggregation/analytics
- “Observability”, according to this definition, is a superset of “monitoring”,
 - providing certain benefits and insights that “monitoring” tools provides



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Data observability

Pravin Y Pawar

Adapted from [What is Data observability? The five pillars](#)
By Barr Moses

Observability

- Observability is no longer just for software engineering!
 - With the rise of data downtime and the increasing complexity of the data stack,
 - observability has emerged as a critical concern for data teams, too.
- Developer Operations (DevOps) teams have become an integral component of most engineering organizations
 - remove silos between software developers and IT, facilitating the seamless and reliable release of software to production
- As organizations grow and the underlying tech stacks powering them become more complicated
 - it's important for DevOps teams to maintain a constant pulse on the health of their systems
- Observability, a more recent addition to the engineering lexicon, speaks to this need, and refers to the monitoring, tracking, and triaging of incidents to prevent downtime.

Observability engineering

- As a result of this industry-wide shift to distributed systems,
 - observability engineering has emerged as a fast-growing engineering discipline
- At its core, observability engineering is broken into three major pillars:
 - **Metrics** refer to a numeric representation of data measured over time
 - **Logs**, a record of an event that took place at a given timestamp, also provide valuable context regarding when a specific event occurred
 - **Traces** represent causally related events in a distributed environment
- Taken together, these three pillars give DevOps teams valuable insights to predict future behavior
 - and in turn, trust their system to meet SLAs

The rise of data downtime

- Common phenomenon
 - Most of the time a report is delivered, only to be notified minutes later about issues with data
 - didn't matter how strong data pipelines were or how many times reviewed SQL: data just wasn't reliable!
- Data downtime tops the list of the problems!
 - periods of time when data is partial, erroneous, missing, or otherwise inaccurate
 - only multiplies as data systems become increasingly complex, supporting an endless ecosystem of sources and consumers
- For data engineers and developers, data downtime means wasted time and resources
- For data consumers, it erodes confidence in decision making

Data observability

- Instead of putting together a holistic approach to address data downtime,
 - teams often tackle data quality and lineage problems on an ad hoc basis
 - much in the same way DevOps applies observability to software
- Data observability refers to an organization's comprehensive understanding of the health and performance of the data within their systems.
- Data observability tools employ
 - automated monitoring,
 - root cause analysis,
 - data lineage,
 - and data health insights
 - to proactively detect, resolve, and prevent data anomalies
- Results in
 - healthier data pipelines,
 - increased team productivity,
 - enhanced data management practices,
 - and ultimately, higher customer satisfaction

Five pillars of data observability



5 PILLARS OF DATA OBSERVABILITY



Freshness: Are your tables updating at the right time?



Volume: Do you have too many or too few rows?



Distribution: Is the value within a normal range?



Schema: Has the organization of the data changed?



Lineage: How are data assets connected across your data stack upstream and downstream?

- Together, these components provide valuable insight into the quality and reliability of data.

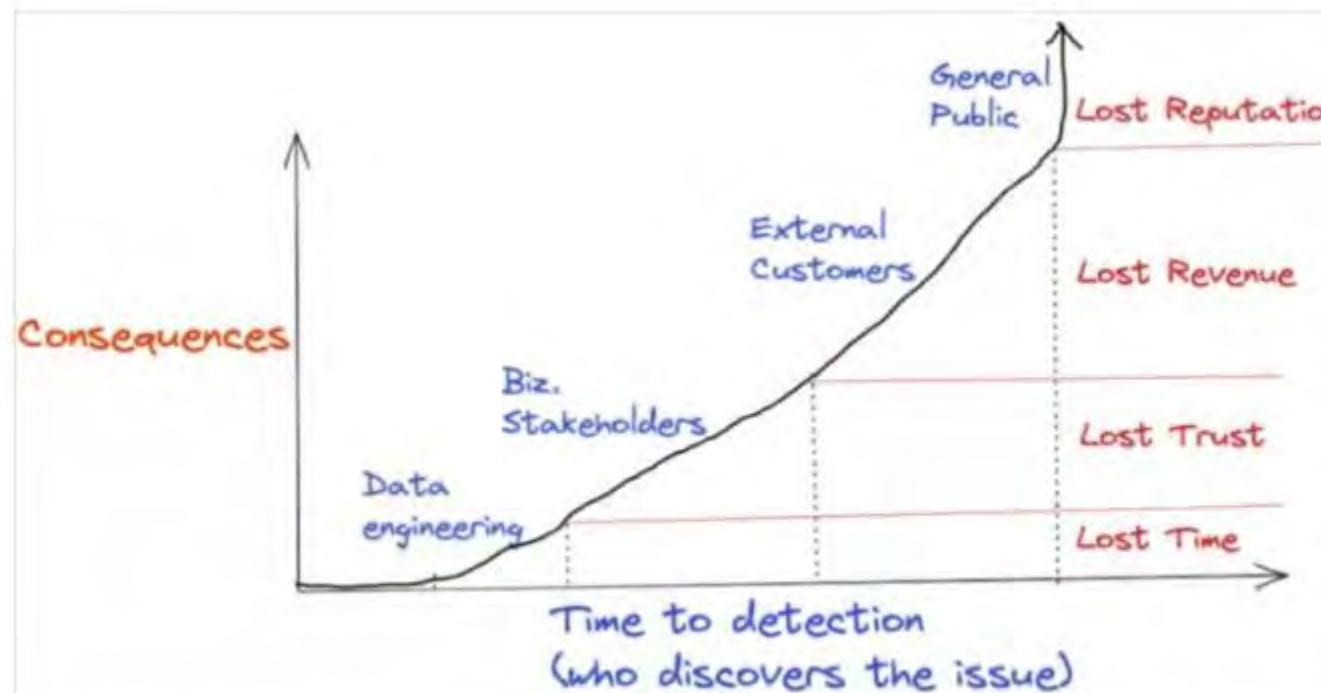
Five pillars of data observability(2)

- Freshness
 - seeks to understand how up-to-date your data tables are, as well as the cadence at which your tables are updated
 - is particularly important when it comes to decision making; after all, stale data is basically synonymous with wasted time and money
- Quality
 - Data pipelines might be in working order but the data flowing through them could be garbage
 - looks at the data itself and aspects such as percent NULLS, percent uniques and if data is within an accepted range
 - gives insight into whether or not tables can be trusted based on what can be expected from data
- Volume
 - refers to the completeness of data tables and offers insights on the health of data sources
 - If 200 million rows suddenly turns into 5 million, should know why

Five pillars of data observability(3)

- Schema
 - Changes in the organization of data, in other words, schema, often indicates broken data
 - Monitoring who makes changes to these tables and when is foundational to understanding the health of data ecosystem
- Lineage
 - When data breaks, the first question is always “where?”
 - Data lineage provides the answer by telling
 - which upstream sources and downstream ingestors were impacted,
 - which teams are generating the data
 - and who is accessing it
- Good lineage also collects information about the data (also referred to as metadata) that
 - speaks to governance, business, and technical guidelines associated with specific data tables, serving as a single source of truth for all consumers.

Why is data observability important?



Data observability is important because the consequences of data downtime can be severe. Image courtesy of Shane Murray.

- For data engineers and developers, data observability is important because data downtime means wasted time and resources
- For data consumers, it erodes confidence in your decision making

The key features of data observability tools

“what are data observability tools?”

- Connects to existing stack quickly and seamlessly and does not require modifying your data pipelines, writing new code, or using a particular programming language
 - allows quick time to value and maximum testing coverage without having to make substantial investments
- Monitors data at-rest and does not require extracting the data from where it is currently stored
 - allows the data observability solution to be performant, scalable and cost-efficient
- Requires minimal configuration and practically no threshold-setting
 - should use machine learning models to automatically learn environment and data
 - uses anomaly detection techniques to let know when things break
 - minimizes false positives by taking into account not just individual metrics, but a holistic view of data and the potential impact
- Requires no prior mapping of what needs to be monitored and in what way
 - helps identifying key resources, key dependencies and key invariants to get broad data observability with little effort
- Provides rich context that enables rapid triage and troubleshooting, and effective communication with stakeholders impacted by data reliability issues
 - shouldn't stop at “field X in table Y has values lower than Z today.”
- Prevents issues from happening in the first place by exposing rich information about data assets
 - so that changes and modifications can be made responsibly and proactively

The future of data observability

- Data observability is a rapidly maturing but still evolving space
 - For example, G2 Crowd created a data observability category in late 2022,
 - but there is not a data observability Gartner Magic Quadrant
 - But Gartner did place data observability on their 2022 Data Management Hype Cycle!
- However, multiple companies and technologies are identifying with the term data observability
 - has been tremendous investor activity, and, most importantly, customer interest and values are at all-time highs
- Data observability is at the heart of the modern data stack,
 - whether it's enabling more self-service analytics and data team collaboration, adoption,
 - or working alongside dbt unit tests and Airflow circuit breakers to prevent bad data from entering the data warehouse (or data lake) in the first place
- A core component of important and emerging data quality best practices such as data mesh, data SLAs, and data contracts



Thank You!

In our next session:

O'REILLY®

Compliments of

KENSU

What Is Data Observability?

Building Trust With
Real-Time Visibility

Andy Petrella

REPORT



Trust What You Deliver

Our low latency data observability solution alerts about data issues, prevents propagation, and highlights which applications are impacted.



Discover how our “Data Observability Driven Development” method is a paradigm shift for data teams that want to scale up their activities without adding complexity to their daily operations.



[Learn More](#)



www.kensu.io

What Is Data Observability?

*Building Trust with
Real-Time Visibility*

Andy Petrella

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

What Is Data Observability?

by Andy Petrella

Copyright © 2022 O'Reilly Media Inc. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://oreilly.com>). For more information, contact our corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com.

Acquisitions Editor: Jessica Haberman

Proofreader: Elizabeth Faerm

Development Editor: Jeff Bleiel

Interior Designer: David Futato

Production Editor: Kate Galloway

Cover Designer: Randy Comer

Copyeditor: nSight, Inc.

Illustrator: Kate Dullea

February 2022: First Edition

Revision History for the First Edition

2022-01-26: First Release

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *What Is Data Observability?*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

The views expressed in this work are those of the author and do not represent the publisher's views. While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

This work is part of a collaboration between O'Reilly and Kensu, Inc. See our [statement of editorial independence](#).

978-1-098-12096-2

[LSI]

Table of Contents

1. It's Time to Rethink Data Management.....	1
Why the Lack of Confidence Can Jeopardize the Future of AI	3
Inadequate Data Management Is Inhibiting Data	
(Application) Innovation	6
2. What Is Data Observability—and Why Do We Need It?.....	9
Characteristics of a Data Incident	11
Achieving Data Observability	13
Using Circles of Influence to Reduce Circles of Concern	17
How Data Observability Influences Team Dynamics	20
Applying DevOps Practices to Data	23
3. Conclusion: The Benefits of Data Observability.....	27

CHAPTER 1

It's Time to Rethink Data Management

In 2016, Microsoft deployed an AI chatbot named Tay on Twitter. Tay had to be shut down only 16 hours after it launched because a number of its nearly 100,000 tweets mimicked offensive and controversial language used by Twitter users that the bot was supposed to be learning from.

Peter Lee, corporate vice president of Microsoft Healthcare, posted the following apology: “We are deeply sorry for the unintended offensive and hurtful tweets from Tay, which do not represent who we are or what we stand for, nor how we designed Tay.”¹

This is an example of how badly a brand, even a leader in its industry, can be hit when data-based decisions go “freestyle.” It also shows how a simple data issue, such as missing or incomplete data, can have significant repercussions.

With the tremendous growth in the volume of data that organizations are collecting, and the scale of its usage, data issues are occurring more often. By 2025, it’s estimated that the global volume of data will expand to 180 zettabytes, more than double the volume of data in 2020. Fueling this dramatic growth is the fact that almost every function of every organization now generates data. It’s also

¹ Peter Lee, “Learning from Tay’s Introduction,” March 25, 2016, Microsoft Corporation, blog post, <https://blogs.microsoft.com/blog/2016/03/25/learning-tays-introduction>.

being driven by the advancement of open source machine learning, which relies on large datasets for training models.

To manage this exploding data growth, data teams have risen in importance and size. Small data teams with few stakeholders have been replaced by large teams that must answer to executive pressure. Yet, despite the increased value and presence data has within organizations, little thought has been given as to how to monitor the quality of the data itself.

While IT and DevOps have numerous quality control measures (such as development practices like continuous integration and testing) to protect against application downtime, most organizations don't have similar measures in place to protect against data issues. But just as organizations depend on high levels of reliability from their applications, they also depend on the reliability of their data.

When data issues—such as partial, erroneous, missing, or inaccurate data—occur, the impact can rapidly multiply and escalate in complex data systems. These data incidents can have significant consequences and multiple negative repercussions on the business, including lack of trust and loss of revenue. A lack of control or visibility into the data quality can also produce faulty insights, which can lead the organization to make poor decisions that can result in lost revenue or a poor customer experience.

Resolving data incidents can be time-consuming and difficult. Identifying the root cause of this type of failure requires knowing where the data flowed from and, therefore, what applications were participating in creating value from that data. But early identification of data issues, which are unknown to the data owners in most cases, is inherently challenging in complex data systems. IT and data teams must become archeologists, trying to discover what's gone wrong. Yet each team is siloed and doesn't have complete visibility. The further downstream the issue, the harder it is to figure out what happened and the more information gets lost in translation. Patches may fix the issue, but if the root cause isn't identified, there's little confidence that the issue won't reoccur. This creates a lack of trust in the data and in the team responsible for shepherding that data to the data user. This, in turn, mutes the data's value.

The lesson? No matter how vast an organization's data assets are or how advanced the technologies are that support the management and analysis of data, they cannot be useful without reliable data

quality. Organizations must pay more attention to data collection, storage, and preparation practices before data analysis takes place. Indeed, the unmanageable size and quantity of data create significant challenges, but you can confront and tackle these issues if sound data management practices are in place.

The practices that will create visibility and add the monitoring capabilities to deal with such challenges are what composes data observability. We'll cover it in more detail in [Chapter 2](#), but before that, let's review together the impact of the status quo in the long run.

Why the Lack of Confidence Can Jeopardize the Future of AI

AI is one of the fastest-growing and most popular data-driven technologies in use. Nine in ten businesses currently have ongoing investments in AI.² Moreover, 86% of companies say that AI will be a “mainstream technology” at their company by the end of this year.³

However, a contradictory “2020 Big Data and AI Executive Survey” also showed that there has been a 39.7% decline in organizations that say they are accelerating their AI investments.⁴ It’s beginning to look more likely that businesses will start to pull back on investments in AI if better returns aren’t seen soon.

Because AI depends upon datasets to feed its models, the reliability of the data can directly impact the success of AI models and, more broadly, the innovation and progression of AI. The issue isn’t that these AI models need *more* data to be successful but that they need *high-quality data*. Even then, a high-quality data model can fail simply because the world has changed too much for the model, which based its learnings on a previous state of the world and therefore can no longer work efficiently under the new conditions.

² “NewVantage Partners Releases 2020 Big Data and AI Executive Survey,” Business Wire, January 6, 2020, <https://www.businesswire.com/news/home/20200106005280/en/NewVantage-Partners-Releases-2020-Big-Data-and-AI-Executive-Survey>.

³ “AI Predictions 2021,” PwC, October 2020, <https://www.pwc.com/us/en/tech-effect/ai-analytics/ai-predictions.html>.

⁴ “NewVantage Partners Releases 2020 Big Data and AI Executive Survey.”

For instance, if a retailer's ecommerce store's target audience suddenly switches from teens to pregnant women because it started carrying several lines of maternity clothes, the AI model might not be capable of predicting the right recommendations for site visitors anymore because it is essentially a new population. So while the data itself might be correct, the real-world circumstances have changed.

Still, data quality is always a precondition for AI. Garbage in will produce garbage out if we don't pay attention to how the outputs are created by the application. A better approach is for the application to detect bad-quality data (garbage in) when it's being inputted into the application to avoid producing a poor-quality data output (garbage out).

According to Roger Magoulas and Steve Swoyer's "The State of Data Quality in 2020 Survey," over 60% of enterprises see their AI and machine learning projects fail due to too many data sources and inconsistent data.⁵ Even small-scale errors in training data can lead to large-scale errors in the output. Incomplete, inconsistent, or missing data can drastically reduce prediction accuracy.

Think of the impact of prediction degradation on something like self-driving cars. A poor prediction, such as not accurately predicting the car's proximity to a human crossing at a crosswalk, could lead to a fatal accident. Something similar has already happened. A Tesla Model S, being operated in full self-driving (FSD) mode, missed a curve in the road, causing it to hit a tree and kill two people. Uber's testing of self-driving technology has also resulted in some grim outcomes. Most recently, an Uber self-driving car killed a pedestrian crossing the road. These types of prediction-failure incidents have created considerable skepticism that self-driving cars will ever be safe. In the case of Uber, it has led to the company halting the testing of the technology in Arizona, where the accident occurred.

If organizations don't implement better quality control mechanisms—and at a scale that can keep pace with the ingestion of massive volumes of data—the risk of failures within AI-driven applications will become too great for organizations to bear. CEOs will begin to say that while the promise of AI was great, it hasn't

⁵ Roger Magoulas and Steve Swoyer, "[The State of Data Quality in 2020](#)," O'Reilly Media, February 12, 2020.

lived up to expectations. They'll begin to believe there's too much at stake in deploying new products or services that rely on AI because they won't feel they can control the quality of the decisions. Companies can avoid the occurrence of this reaction by filling the data quality control gap with data observability.

For example, big data is eventually seeing its hype cycle era coming to an end (see [Figure 1-1](#)). Nearly three-quarters of organizations cite big data as an ongoing challenge, and only 37.8% say they've been able to create a data-driven organization. One positive aspect of this shift away from big data is that there's now less focus on collecting large volumes of data and more emphasis on the challenges of processing, analyzing, and interacting with massive amounts of data.

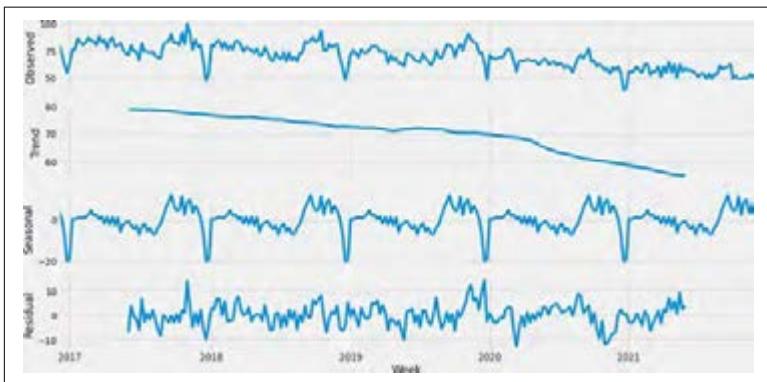


Figure 1-1. Timeseries analysis performed on Google Trends for searches of “big data” worldwide

We are at a pivotal point in the AI hype cycle. If AI is to avoid the fate of other AI hype cycles, its success must continue to impress. By impress, I mean that AI must continue to be simultaneously creative (new and innovative) and performant (efficient and trustable). For both conditions to occur, as I'll discuss in more detail later on, quality data assurance is a must.

Inadequate Data Management Is Inhibiting Data (Application) Innovation

Data systems are already complex. The data being used to power AI models and ML technologies is beyond the scope of human capabilities to process. This fact raises ethical and practical concerns when building new data applications.

While there are humorous instances of AI failure, such as an AI-powered camera mistakenly tracking a soccer player's bald head instead of the ball, the reality is there are many situations where AI could cause actual harm. There have been instances of AI applications that violate privacy laws, show inherent gender and racial discrimination, and promote disinformation. But AI's potential for malicious use doesn't stop there.

Some of the world's leading technology companies are already running up against this dilemma. An artificial intelligence engine created by Facebook, for example, was shuttered because the AI had made its own unique language that humans couldn't understand.

As these use cases highlight, if you can't control the quality of both the inputs and outputs, the application is also at risk of being uncontrollable. If this becomes the case (or even if this becomes the prevailing sentiment about AI-powered applications), many organizations and developers will feel it's necessary to limit themselves to less complex AI applications to reduce risk. However, this will also restrict their ability to be creative and innovative with the technology.

Another way data application innovation could become limited in the future is due to regulatory concerns. Data privacy regulations like General Data Protection Regulation (GDPR) and California Consumer Privacy Act (CCPA) must be followed. But at the same time, regulatory concerns can dampen experimentation and innovation. For instance, if a developer wants to use consumer data in an application but doesn't know exactly how they want to use it, access to that data may be restricted. The data owner won't want to share the data because they bear ultimate responsibility for that data. If the data is used in a way that doesn't comply with data regulations, they'll be held responsible.

We can already see examples of this type of situation occurring. For instance, when Facebook shared personal user data with Cambridge Analytics, it endured significant blame and backlash—perhaps even more so than Cambridge Analytics—for how Cambridge Analytics misused that data. Yet Facebook did not have prior knowledge that this was how the data would be used. Having seen what happened to Facebook, other data owners have become much more concerned about the implications of sharing their data with third parties. These concerns threaten to limit developers' ability to experiment with and innovate new data applications.

One workaround for data owners is to sample and anonymize the provided data to alleviate risk. But an anonymized sample with some of the data stripped out doesn't replicate the real world, leaving the analyst with little confidence about how that application will perform in production.

An analogy here is a tightrope walker. When the tightrope walker is first attempting to tightrope walk, they need the security of a net beneath them because the likelihood of falling is high. Data owners and developers also need the security of knowing that they have a safety net to help them identify data quality issues and remain in compliance with data regulations. Otherwise, they will limit the complexity of what they build in order to limit their own risk.

This fear of losing control of the data quality as complexity increases doesn't just limit data application innovation; it also limits AI and ML creativity and innovation because these are also data-driven applications.

Because there are significant negative repercussions possible from AI, including revenue loss, reputational damage, regulatory penalties, and a lack of public trust, organizations must proceed with caution. So how do you manage the complexity of data-driven AI systems? How do you ensure the quality of the data so you can mitigate risk? How do you build trust in that data so you can have confidence in the outputs of the AI algorithms?

To answer those questions, the fundamental challenge is how developers can get timely insight into a system's state, especially when complex cloud-native ecosystems built upon massive data volumes make it much more difficult to predict a system's behavior. To address this challenge, there's been a growing movement toward implementing observability to gain deeper contextual insight into

the correctness and performance of these complex data systems. With regards specifically to data management, data observability has emerged as a tool that can provide the visibility and safety net necessary to spur greater data-driven innovation.

In the next chapter, we'll dive deeper into what data observability means and how it can be applied to complex data ecosystems.

CHAPTER 2

What Is Data Observability—and Why Do We Need It?

In your organization, has someone ever looked at a report and said the numbers were wrong? Likely this has happened more than once. No matter how advanced your data analytics and modeling tools are, if the data you’re ingesting, transforming, and flowing through your pipelines isn’t correct, the results won’t be reliable.

Even one new field added to a table by one team may cause another team that relies on that same data, but for a different use case, to have inconsistent data. If this issue isn’t quickly identified, the downstream impact could result in compliance risks, poor decision making, or lost revenue. If it happens too often, it will also result in a severe loss of confidence by business users.

However, discovering these kinds of data failures is much more challenging than typical application or system failures. In the case of an application, when something isn’t working, the symptoms are more evident. For instance, if an application crashes, freezes, or restarts without warning, you know you’ve got a problem. However, data issues are generally hard to notice since the data won’t freeze, crash, restart, or send any other signal that there’s a problem.

At the usage level, data also can’t just be “fixed.” You can only correct it by requesting the producer to publish the fixed data or by looking at the application(s) producing the data to determine what’s gone wrong. For instance, if a column or row is missing when the data is ingested, you can’t just fix the data. In this specific case, the data

isn't there at all, so fixing the data requires fixing the application upstream to ensure it ingests the entire dataset. In another case, the data may simply no longer be available and, therefore, the producer of that data needs to be made aware so that they can adapt their analysis to this new reality. Further, when you're performing an analysis of the data, certain assumptions are made that may no longer be true at a later point in time. Here's an example of what I mean.

When an analyst initially builds a model to analyze a retail organization's previous year's sales data, they can see that the organization collected data every month for the prior year. Thus, the model is trained based on a period of 12 months of data from the last year. However, while the data was complete when the data model was initially built, that may no longer be the case. Perhaps, when the analysis is deployed six months later, the assumption of having the 12 months of data (with the seasonality of four quarters) is no longer true because the system can no longer handle this amount of data. As a result, the previous year's data is removed from the analysis. Since Q4 tends to be some of the highest-grossing months for a retailer, this change can have a major impact on whether the insights generated from the data are accurate, especially when the assumption is that this data is present. However, because the analyst hasn't shared their assumptions with the producer and the producer doesn't have that level of visibility into the data models, they are not aware of this "implicit" constraint coming from the data usage. Hence, they also don't know that the output from the model (their analysis) is inaccurate.

This is where data observability comes into play. You need to be able to observe and be aware of these types of silent changes to the data so that you can fix them preemptively—before your CEO is coming to tell you that numbers look wrong.

Data observability is a solution that provides an organization with the ability to measure the health and usage of their data within their system, as well as health indicators of the overall system. By using automated logging and tracing information that allows an observer to interpret the health of datasets and pipelines, data observability enables data engineers and data teams to identify and be alerted about data quality issues across the overall system. This makes it much faster and easier to resolve data-induced issues at their root rather than just patching issues as they arise.

Critically, what makes a data observability solution unique from application observability is that the data must be logged and traced from within the data pipelines, where the data is created and activated for use. This allows you to measure the pertinence of the usage of your data within your entire system (all your applications and pipelines) as well as monitor health indicators of the overall system.

Why is this important? Because while your data pipelines may look fine—the data isn't using too much memory or taking up too much storage space—if the data outputs from those pipelines are providing garbage data, then the value of the data is worthless. On the other hand, if you only observe a table within your database, it may tell you what queries were performed, but you won't know how it's being used, or by whom. This matters because you won't know if the data being used is of value to the end user.

Thus, to ensure that the data analytics you're performing are accurate and valuable, you need to observe both the data and the pipelines at the same time. If you're unable to have this type of end-to-end visibility, you're likely to suffer what I call a *data incident* or a *data failure*. In the next section, we'll dig deeper into what this means and why they should be avoided at all costs.

Characteristics of a Data Incident

Data incidents result from a missing gap in data management that leads to catastrophic impacts on businesses. In an increasingly complex system of data pipelines, applications, and numerous inputs of large volumes of data, there has been a proliferation of such incidents recently. So what causes them?

The cause could be inaccurate, incomplete, or inconsistent data. Or it may be that data is not being refreshed at the right timing interval, so that when the business user needs to pull insights from the data, the data hasn't been updated yet. Or the data might be refreshed on time, but it's too old to be valid (i.e., the data may have just been refreshed, but if it's pulling the previous month's sales numbers and not the current month's numbers, it's incorrect).

Assumptions that were made earlier in a data experiment may not be true anymore. Perhaps the age parameters have changed in the data, so the previous assumption that all data will be from people aged 65 and older is now incorrect. Or the data could also be

out of compliance and therefore unusable. For instance, you can't use personally identifiable information (PII) from a health-care provider database to predict the next election results. This data may be accurate, but it's illegal to use health PII data in this manner.

Data issues can also occur when an application has been altered because it has received a request from a business user to change the way data is transformed. While this change may be helpful for one business user, the change may impact other users who have a different need for perceiving and using the data. So even if the data is published in the same format and at the same time, if some of the data structure has changed, the data may no longer be relevant for certain use cases. What's more, the impacted business users (those who didn't request any change to the data) also won't have the visibility to know that the structure of the data has changed. As a result, they may be producing incorrect analyses from the data.

In a large organization, even one minor change somewhere in the dataset or data pipeline can create a snowball effect, where the change upstream creates a bigger and bigger problem as it moves downstream, making the impact much greater. Inaccurate data can either be injected into the system at the time of collection from a third party, such as a customer who submits an incomplete form, or during the transformation process. For instance, a data engineer may have been asked either to remove incomplete fields (those with too many nulls) or to fill them with approximations. However, by making this change to the business rule, the data is now corrupted for another user with a different use case. Thus, the application produces incorrect insights, which leads to poor decision making from the business because the data injected was inaccurate.

As you can see, there are multiple steps in the data process, and at any step, the data can be corrupted. By corrupted, I mean merely that the quality of the data is not adequate for how it's going to be used. Data itself cannot be wrong or inaccurate, but its usage becomes inadequate for a particular use case. Thus, the earlier you are able to intercept a data adequacy issue, the less damage there will be and the higher the loss prevention.

When considering the explosive growth of AI and the complexity in these models and applications, incidents can be nearly impossible to resolve at the root cause without visibility into the data and data pipeline. But if issues aren't resolved adequately and continue to

reoccur, it will ultimately lead to a lack of confidence in the ability to create complex data-driven applications.

Knowing that data observability is necessary to avoid a drawdown in confidence in using large datasets and AI-driven applications, I'll now discuss how to achieve data observability.

Achieving Data Observability

We've discussed the necessity for data observability and the need to observe the data along its usage within the systems. But, in practice, how does data observability work?

Similar to DevOps observability, which relies on automated monitoring to identify leading indicators of service degradation or unauthorized activity, data observability also relies on automated logging and tracing of the data and data pipelines to evaluate data quality and identify issues.

Assessing Data Quality

There are many indicators of data quality¹ to consider. Here are some of the most usual ones:

Accuracy

The data values should be reliably representative of the information (e.g., phone numbers should be real phone numbers).

Completeness

The data should contain all the necessary and expected information (e.g., a complete address would contain the street address, city, state, and zip code).

Consistency

Data should be recorded in the same manner across all systems (e.g., phone numbers should all be collected either with or without the country code, rather than a mix of some with country codes and some without).

¹ "Data Quality," Wikipedia, last updated November 16, 2021, https://en.wikipedia.org/wiki/Data_quality.

Conformity

Data should be collected in the same format (e.g., phone numbers should only have numbers, not letters, and email addresses should always have an @ sign and end in .com).

Integrity

The data must be connected to other data across all relationships and not be considered as an orphan (e.g., if you have a phone number in your database, but it's not associated with a person, the data is invalid).

Timeliness

Data must be up to date, and it must be ingested at the correct intervals (e.g., if you are analyzing weekly point-of-sale [POS] data, the data should be refreshed weekly).

Unique

Data needs to be cleaned of duplicate entries (e.g., if you have two entries, one with the name Jane Smith and another with Jane G. Smith, and both correlate to the same individual, one record should be deleted).

Observing Data Quality Within Your Data Systems

Now that we've defined what constitutes data quality, the next step is to ensure that these quality indicators are all present within your datasets and systems. Using principles similar to software application observability and reliability, and applying it to data quality, these issues can be identified, resolved, and proactively prevented.

To generate this type of end-to-end data observability, it's necessary to log and trace the following information within your data pipelines and datasets:

Application

You need to log the application that is using or creating the data. This activity is often wrongly considered part of logging and tracing the lineage. However, lineage activity shows the links between the data independently of the application, so you must also log and trace the application.

Users

It is important to log who created, maintained, and ran the applications interacting with the data. This way, you can quickly

identify who can help with data issues and resolve the matter faster.

Lineage

Logging lineage allows you to map the connection between applications and data sources (i.e., the data flow) so you know which applications are generating the data and who is accessing or using that data. By tracing the lineage when an application changes or its outputs become erroneous, you can identify what data and data users are impacted.

Distribution

Distribution refers to the shape of the data. It can be the number of occurrences of categories or the descriptive stats (e.g., average, minimum, quantiles, etc.) of numerical values. It is important to know the distribution of values to match their relative adequacy to their usage. If data distribution varies too much, it's an indicator that the initial assumptions may be wrong, which in turn means the analysis of the data may be inaccurate.

Time-based metrics

There are several forms of time-based metrics that you should log and trace. These include:

Frequency

Logging the pace at which the data is updated allows you to determine whether it is being updated at the appropriate cadence.

Freshness

Logging when data is created, updated, or deleted allows you to determine how "fresh" the data is. This will then allow you to identify whether it's up to date or if it needs to be refreshed.

Time frame

Logging the length of time the data spans (e.g., the time frame could be five years, a year, or only a week of data) allows you to determine whether the data is complete.

Completeness

The data must be complete to ensure the maximum information available is at play. There are two dimensions to consider:

Missing values

Logging missing values indicates that no data value is stored for the variable in an observation. Knowing that you have a missing value is important because missing values can add ambiguity and result in an inaccurate analysis of the data.

Volume

Logging the amount of data used or created by an application allows you to determine whether your data generation is meeting expected thresholds and determine the completeness of your data. For instance, if your data volume suddenly changes from 500 million rows to 5 million, you'll want to take a closer look to determine why.

Schema

Schema changes occur when fields or tables are added, removed, or changed. Logging schema changes will help you with ensuring the accuracy, completeness, and consistency of your data.

In addition to monitoring these events, a robust and holistic approach to data observability also requires observing data usage within both the pipelines and the overall data systems. Thus, I'm emphasizing once again (because it's such a critical point!) that you want your applications to create logs for both data usage and the applications within the data pipelines or the overall systems.

Finally, when measuring data quality, you need to think about the differences between input and output (i.e., what inputs you need to get an accurate output). One way to approach this problem when you have massive datasets is to create metrics only around the data the application consumes. For instance, if you have a database with 10 billion entries, but your application only uses 1 million of those entries, then you need only log and trace what you care about within the 1 million entries. So, if the 10 billion entries include data from all countries globally in this instance, but your use case only relates to the 1 million entries from Russia, you should determine what your data quality key performance indicators (KPI) are specifically for those 1 million Russian entries. Those KPIs might be tracing whether you received too many missing addresses, whether you're getting the same number of fields in a table, whether you still have the same number of entries over time, etc.

Whatever your KPIs are, by setting metrics for your data quality and then logging events across all systems, you have the visibility to observe whether those data quality metrics are being met. And when they're not, through tracing, you can discover what's causing the error. This combination—metrics, logging, and tracing—provides true end-to-end visibility across all pipelines and data.

Now that you understand what is required to create data observability within your data pipelines and datasets, we'll move on to discussing the more practical issue of how to manage data issues that are within your area of responsibility and those outside it.

Using Circles of Influence to Reduce Circles of Concern

When we implement data observability and data quality, it's important to recognize the full complexity of the situation within an organization. When it comes to data management, there are often issues outside our control that impact the data quality for a particular use case. This can lead to high levels of frustration.

Acknowledging that we all have challenges outside our power to control, Stephen Covey developed a framework called the “circle of influence and control” in his book, *The 7 Habits of Highly Effective People* (Free Press). Here's how he defines each circle in the framework:

Circle of concern

Challenges or issues that have a direct impact on you

Circle of influence

Areas within your life where you can indirectly control or influence the outcome

Circle of control

Areas within your life where you have direct control of the outcome

As Covey points out, the challenge is that there are a lot of issues within our circle of concern that we do not have direct control over. We cannot directly, as individuals, eliminate many of the concerns we have that impact our quality of life, such as the high cost of health care. However, there are ways we can indirectly influence these areas of concern. For instance, we can use our influence to eat

healthier and exercise in order to reduce the likelihood of needing to use health-care services. Or, if we are concerned about the high cost of gasoline, we can purchase an electric vehicle to avoid paying for gas altogether.

Let's take this circle of concern/influence/control framework and apply it to data management within the enterprise. We can see that in the world of data there are often circles of control and influence and circles of concern:

The circle of concern

The area or scope where data issues can create problems, but you have no power to fix them

The circle of influence

The area or scope of data issues within which you have the capability to influence a resolution of the issue, if not directly fix it

The circle of control

The area or scope of data issues that you can directly fix or resolve

To give an example of how this might apply to a real-world scenario, let's look at a pharmaceutical company that ingests data from multiple external labs and uses that data to determine what molecules or compounds might be effective in developing a new drug. The pharmaceutical company has an application to help them with this process and specific data inputs needed to conduct the analysis. This application, which was built by the pharmaceutical company, is within its circle of control. The pharmaceutical company has complete power to make changes to the application. However, the data that comes from their external labs and is ingested by the company's application is outside of its circle of control. If that data is incomplete or inconsistent when the application ingests it, it will produce a faulty analysis. The problem is that this issue falls into the circle of concern—an area where the pharmaceutical company is impacted by the issue but has no direct way to control the outcome (e.g., identify and fix the data issue).

However, through data observability, the pharmaceutical company can still use its circle of influence to help resolve data issues within its circle of concern. By logging and observing the data quality within its own application, the pharmaceutical company can identify

the specific lab the data issue is coming from, and it can even further pinpoint what data is creating the issue. So, while the company can't fix the problem itself because it's outside the company's circle of control, it can use its circle of influence by sharing its findings with the affected lab to help speed a much faster resolution to the concern. Additionally, this level of observability allows the pharmaceutical company (the consumer) and the labs (the producers) to create mutually acceptable service level agreements (SLAs) that can validate from both the producer and consumer perspective that the data matches expectations and meets contractual requirements. This is creating a culture where teams are sharing concerns and combining their influence in order to face issues together, as represented in Figure 2-1.

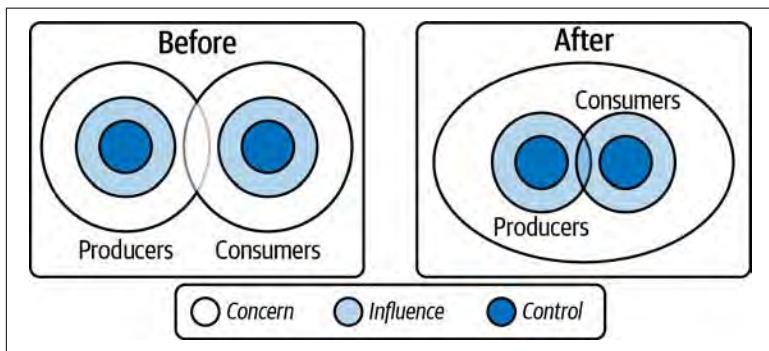


Figure 2-1. With data observability, circles of influence expand and enable stronger collaboration

Without data observability, not only would the data quality issue be outside the pharmaceutical company's control, but it would also be very difficult and time-consuming even to pinpoint which external input (lab) was creating the problem. And even once the company could identify the lab causing the issue, without observability into the data itself, it still would not have a clear understanding of the root cause. But, by being able to trace the issue within its own systems and datasets, the company provided a much more informative and detailed report of the issue to the lab. In turn, this improves the affected lab's ability to resolve the issue and speeds the time it takes them to do so. Thus, using circles of influence, you can observe and identify data quality issues, build greater trust in the entire system, and find and fix the root cause of a data issue more quickly.

Next, let's look at how data observability facilitates teams working together.

How Data Observability Influences Team Dynamics

Another area of complexity regarding data management within an organization is how roles and responsibilities focused on managing and using data are structured. Within any data-driven organization, there are several teams involved in the use of data. The exact structure and types of teams may differ in every organization (e.g., the analytics teams might be split across the data and business teams in some organizations), but there are typically four primary teams involved in managing, analyzing, and utilizing data. Each team has different responsibilities as well as different dependencies on other teams to ensure they can manage their responsibilities ([Table 2-1](#)).

Table 2-1. The teams

Type of team	Responsibilities	Dependencies
IT team/ production and operation team	Build and maintain the platform; manage security and compliance; oversee site reliability engineering (SRE).	Receive recommendations from other teams about what to monitor and what to consider as an error.
Data team	Build the data pipelines and manage the data from deployment to production.	Rely on IT to build the platform and set up the tools to monitor the pipeline and systems, including data observability tools.
Analytic/data science team	Build analytics and AI/ML models that can analyze and interpret data for the business team's use cases.	Rely on the data team to build the pipeline to generate the data it will be using in its models.
Business/ domain team	Sponsor use cases and use data analysis to make business decisions.	Rely on the other teams to ensure data and analyses are accurate.

While this structure allows each team to maintain its own quality controls and outcomes, this process also creates silos. Each team has limited visibility into the functions of the other teams and how the data or pipelines might need to be changed (or not) to fit those functions. This makes it difficult for the IT team to anticipate any changes and take preventative measures to avert any issues experienced by the data or analytic teams.

As a result, the data and analytic teams are alerted to potential issues either too late or not at all. Inevitably, this causes failures in their areas of responsibility, resulting in inaccurate insights being delivered to the business team. Without visibility, it's also very hard to find the root cause of an issue. Instead, the IT or data team might uncover a proximal cause, which they patch. But because the root cause hasn't been discovered, there's little confidence that the same issue won't occur again.

Another aspect influencing team dynamics within organizations is the widespread embrace of a data architecture built on cloud data lakes. The real-time availability and streaming this cloud-based architecture provides is intended to make managing large volumes of data from multiple sources easier and faster to ingest, transform, and serve up. However, this type of architecture often gives teams less control over the increasing volumes of data, resulting in less accountability, which leads to a backlog of unresolved data issues. Consequently, the business team begins to lose trust in IT and data teams. As a result (not surprisingly), a culture of shadow IT emerges as analytics and business teams seek to bring more confidence and control into the data analysis process by internalizing some IT resources within business departments instead of relying on the existing IT department.

What can be done to resolve this negative feedback loop that leaves no one within the organization fully accountable and responsible for resolving data issues? An emerging approach that is beginning to gain wider acceptance is the use of a data mesh.

Let's take a closer look.

Using a Data Mesh to Increase Team Accountability and Responsibility

As discussed earlier, the typical organizational team structure for managing data has led to accountability and responsibility issues—each team is siloed and not accountable for issues that arise upstream or downstream from its area of responsibility. The data mesh—an organizational construct that supports data-driven organizations by leveraging a distributed, domain-specific, self-serve design—helps alleviate these silos by increasing accountability and facilitating cross-collaboration communication.

Developed by Zhamak Dehghani, the director of emerging technologies at Thoughtworks, the data mesh is “a distributed data architecture, under centralized governance and standardization for interoperability, enabled by a shared and harmonized self-serve data infrastructure.”²

While a data mesh doesn’t introduce data quality standards, it provides standard guidelines to enable better management of data products, including their quality. These underlying standards (such as standardizing on governance, discoverability, formatting, etc.) help facilitate cross-domain collaboration, especially when data is important to more than one domain. As Dehghani explains in her book, a data mesh has an architecture similar to what is shown in Figure 2-2.

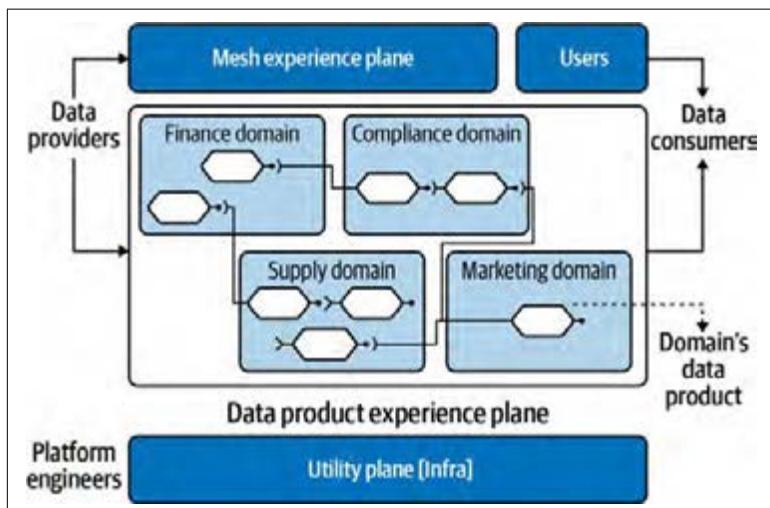


Figure 2-2. Data mesh planes supporting the domain’s data products and their usage³

The data mesh’s universal standard of data governance distributes data ownership among domain data owners responsible for providing their data as products. It also ensures a standardized and scalable

² Zhamak Dehghani, “How to Move Beyond a Monolithic Data Lake to a Distributed Data Mesh,” May 20, 2019, <https://martinfowler.com/articles/data-monolith-to-mesh.html>.

³ Adapted from an early version of Zhamak Dehghani’s book, *Data Mesh* (O’Reilly, 2022).

way for individual domains to handle agreed-upon data quality, such as:

- Data discovery and versioning
- Data product schema
- Data governance
- Data standardization
- Data lineage
- Data monitoring and logging

As a result, teams can apply observability to the data to determine whether their data is fresh, complete, correctly distributed, and so on. In many ways, the data mesh is like the data platform version of microservices: each domain handles its own data pipelines with defined and agreed-upon SLAs that they guarantee to users.

Because data mesh principles primarily relate to accountability, transparency, and responsibility, there is shared visibility into the data and pipelines. Thus, it becomes much easier for both teams to discern their circles of influence. As a result, problems can be more easily identified, and IT and data teams can work together to fix issues upstream before they create downstream issues. And, with more time freed up, data teams can focus more time on data innovation.

Because of three factors—the rapid growth in the number of pipelines, the increase in the number of people managing the pipelines, and the introduction of complex AI/ML models—many of the data issues we've been discussing have compounded. Therefore, in the next section, we'll explore how increased transparency (or observability) can be applied to organizations to allow them to scale data quality management.

Applying DevOps Practices to Data

The concept of monitoring key business processes has existed for quite some time. In the IT and DevOps environment, due to the need for greater visibility, confidence, and speed in their applications and systems, those teams developed processes around their release cycles to avoid errors and dependency failures. These processes were initially manual, but as organizations scaled from a few

developers to thousands, it became necessary to automate these processes.

With the almost universal adoption of the cloud, IT introduced continuous integration/continuous deployment (CI/CD). Implementing CI/CD has allowed the IT and DevOps teams to automate the integration of code changes from multiple contributors into a single software project and then test and validate that the new code won't break the application. By adding automation to the process, IT and DevOps teams vastly improved their ability to deploy applications confidently and quickly. Automation has allowed them to:

- Reduce the overhead of manual coordination
- Reduce the amount of manual work
- Let developers experiment fearlessly
- Create automated tests
- Increase confidence in each release
- Find out sooner when something breaks
- Automate deployments
- Centralize monitoring

We're now seeing a similar recognition within organizations for visibility, confidence, and speed at the data level, not just the application and systems level. Just like businesses can't afford application downtime, they've also come to realize that they can no longer afford issues with the data because they're using it to make strategic decisions.

In the past, there hasn't been a way to automate the testing and validation of data in the same way there is in the IT and DevOps environment. Thus, a data issue can be very time-consuming and difficult to resolve because it requires a lot of manual coordination and work.

For instance, an ecommerce platform may be using complex machine learning models to provide personalized recommendations to site visitors. Data from many different business segments are piped into this model to train it—customer relationship management (CRM) data, POS data, third-party partner data, contact center data, and so forth. But if that data breaks at some point from when it's created to when it's fed into the model (perhaps some of the

demographic fields from the CRM, such as the age or gender, are incomplete), the training models will be incorrect. In this example, customers might start seeing recommendations that are completely misaligned with their interests, such as a 40-year-old male receiving a recommendation to purchase a dress designed for girls between the ages of 6 and 12.

Suppose the IT team of the ecommerce platform conducts a manual investigation to determine why its personalized recommendation engine is broken. The team may find that it's almost impossible to find the root cause, but it does discover a proximal cause. In this instance, the numbers of missing genders and ages in the dataset have increased. Therefore, other features of the person have taken precedence over age and gender, which leads to poor quality recommendations as these two variables are known to have a major influence on the quality of the model. The result is that the distribution of visitors' ages has varied to the point that the application can no longer manage it. This "garbage out" data is then consumed by the marketing automation tool that sends the recommendations to consumers, thus creating a garbage in, garbage out scenario.

However, simply patching the issue doesn't resolve the root cause, which may have possibly been due to an error occurring during the profile processing. This leaves the organization in a precarious situation. Either it must spend significant resources and time trying to uncover the root cause manually, or it can simply fix the proximal cause quickly but risk that the issue could reoccur and cause even more significant issues next time.

This type of issue can be intercepted to avoid the downstream cascade of incidents, but it requires constant checkups and visibility into your pipeline and datasets. A data observability solution is a system that aims at supporting DevOps practices in the context of data and provides the necessary visibility and validation of your data quality, and it does so in an automated fashion. This is similar to how IT and DevOps have deployed CI/CD to give them more visibility and confidence in their applications and systems while also speeding time to market and reducing downtime.

Having observability of your data and data pipelines is important across all data use cases but particularly for AI/ML, where there is also a high level of complexity in the algorithms. If IT and data teams don't have visibility into data and data pipelines, the

probability for data incidents increases significantly. At the same time, the ability to resolve these issues in a timely manner decreases dramatically.

In the next chapter, we'll explore more about how you can use data observability to make your organization data incident-resilient and strengthen AI innovation and data application creativity.

CHAPTER 3

Conclusion: The Benefits of Data Observability

A data observability platform helps IT and data teams detect and stop the propagation of data incidents by tracking and measuring data usage performance across systems, projects, and applications in real time. With a data observability platform, it's much easier to find and fix the root cause because you have:

- Better visibility into how data is being collected, copied, and modified by any application
- The ability to leverage lineage and historical data information to find the initial cause
- The ability to detect anomalies based on historical data information, which is particularly useful in AI/ML applications

In addition, a data observability platform can facilitate better communication across teams and organizations because it provides you with evidence-based information on your data management ecosystem.

You can also use a data observability solution preventatively to observe issues before they occur. You can do this by:

- Logging context and runtime information from within the application code
- Defining quality control rules and thresholds to anticipate data issues

- Validating rules before and continuously in production to generate notifications about specific data events and their context
- Reviewing or refactoring your delivery upon triggered rules

As organizations continue to scale their use of data, they must actively work to become data incident-resilient. Otherwise, data issues will snowball and so too will the negative repercussions to the organization. To achieve data resiliency and avoid data incidents, there are three key steps organizations must take:

- 1. Increase the visibility and awareness of data concerns*

While many IT and data teams may have heard of data observability, it is not currently a common best practice within most organizations. But it must become one. The awareness of data concerns should also go beyond the data engineer all the way to the CIO and even to the CEO (more on this soon).

- 2. Contextually log, measure, and trace data usage*

Logging, metrics, and tracing must also be automated at the systems and data levels, especially within the production environment. Because the data changes, validating code against data change must be part of the development and deployment phases.

- 3. Use automation to continuously validate and test the quality of data in production*

Automation is the only realistic answer at the scale at which today's data-driven organizations use or want to use data. Achieving automation requires also building in data control rules on indicators (or metrics) as part of the development process of every application. This is a new step in the application's life cycle compared to IT's continuous integration and testing process.

With these steps in place, it becomes possible to quickly identify in near real time where data issues are occurring, even proactively, and resolve them at the root issue. Data observability builds greater confidence in the data throughout the organization, allowing it to improve decision making and experiment and innovate more with data applications.

To realize the possibilities and opportunities of AI, organizations must first build a culture that understands the value and importance of proper data management and prioritizes ensuring the uptime of its data through continual data observability. However, only 24% of companies say they've built data-driven cultures.¹ Nearly half (40%) cite a lack of alignment within the organization as a barrier to being data-driven.² Moreover, around 1 in 10 businesses say they don't know where data ownership has sat or will sit in the future.³ This will need to change if organizations want to support better data management and maintain a coherent and cohesive data strategy.

While Dun & Bradstreet reports that there's a growing recognition that responsibility for data should be a priority for the C-suite, there is still uncertainty around who in the leadership team owns data management. The most common answer is that data responsibility lies with the chief executive officer rather than the chief technology officer or chief information officer. In truth, the responsibility lies with all three executives. Cross-organization buy-in of the importance of maintaining data quality is necessary to put the appropriate technologies and processes in place to achieve good data management governance throughout the organization.

Additionally, to fully realize the benefits of AI, organizations will need to build their business practices around replacing a culture of secrecy with one of transparency. While achieving this ideal "data state" is still a distant dream state for many organizations, there are signs that executives see the value in building a stronger data-driven culture, especially to support AI initiatives. In one study, 81% of executives said they were optimistic about the outlook for data and AI within their firms.⁴

1 NewVantage Partners, *Big Data and AI Execution Survey 2021: The Journey to Becoming Data-Driven—A Progress Report on the State of Corporate Data Initiatives*, 2021, https://c6abb8db-514c-4f5b-b5a1-fc710f1e464e.filesusr.com/ugd/e5361a_d59b4629443945a0b0661d494abb5233.pdf.

2 Randy Bean and Thomas H. Davenport, "Companies Are Failing in Their Efforts to Become Data-Driven," *Harvard Business Review*, February 5, 2019, <https://hbr.org/2019/02/companies-are-failing-in-their-efforts-to-become-data-driven>.

3 Dun & Bradstreet, *The Past, Present, and Future of Data*, 2019, https://www.dnb.com/content/dam/english/dnb-data-insight/DNB_Past_Present_and_Future_of_Data_Report.pdf.

4 NewVantage Partners, *Big Data and AI Execution Survey 2021*.

There will be a positive ripple effect as more organizations begin to make the necessary culture shifts. This includes adding more accountability and ownership for the various teams that interact with the data and prioritizing the confidence of data usage through continual data observability, as DevOps and business applications have already done. We'll see data observability acting as the safety net and thus providing greater confidence in complex AI/ML models and applications. Hence, developers will also have more confidence that they can experiment safely with data applications—even highly complex ones. This, in turn, will boost data creativity and innovation, and keep the hype of AI alive.

About the Author

Andy Petrella is an entrepreneur with a mathematics and distributed data background.

In the data community, Andy is known as an early evangelist of Apache Spark and the creator of Spark Notebook. He has also been an O'Reilly author and trainer since 2015, covering topics such as distributed data science, data lineage essentials, data governance, and machine learning model monitoring.

Andy is also the founder and CEO of Kensu.io, a low latency data observability solution that comes with a specific method: Data Observability Driven Development.

Synapse Analytics provides a managed service for large-scale, cloud-based data warehousing. HDInsight supports Interactive Hive, HBase, and Spark SQL, which can also be used to serve data for analysis.

- **Analysis and reporting:** The goal of most big data solutions is to provide insights into the data through analysis and reporting. To empower users to analyze the data, the architecture may include a data modeling layer, such as a multidimensional OLAP cube or tabular data model in Azure Analysis Services. It might also support self-service BI, using the modeling and visualization technologies in Microsoft Power BI or Microsoft Excel. Analysis and reporting can also take the form of interactive data exploration by data scientists or data analysts. For these scenarios, many Azure services support analytical notebooks, such as Jupyter, enabling these users to leverage their existing skills with Python or R. For large-scale data exploration, you can use Microsoft R Server, either standalone or with Spark.
- **Orchestration:** Most big data solutions consist of repeated data processing operations, encapsulated in workflows, that transform source data, move data between multiple sources and sinks, load the processed data into an analytical data store, or push the results straight to a report or dashboard. To automate these workflows, you can use an orchestration technology such Azure Data Factory or Apache Oozie and Sqoop.

Azure includes many services that can be used in a big data architecture. They fall roughly into two categories:

- Managed services, including Azure Data Lake Store, Azure Data Lake Analytics, Azure Synapse Analytics, Azure Stream Analytics, Azure Event Hubs, Azure IoT Hub, and Azure Data Factory.
- Open source technologies based on the Apache Hadoop platform, including HDFS, HBase, Hive, Spark, Oozie, Sqoop, and Kafka. These technologies are available on Azure in the Azure HDInsight service.

These options are not mutually exclusive, and many solutions combine open source technologies with Azure services.

When to use this architecture

Consider this architecture style when you need to:

- Store and process data in volumes too large for a traditional database.
- Transform unstructured data for analysis and reporting.

- Transform unstructured data for analysis and reporting.
- Capture, process, and analyze unbounded streams of data in real time, or with low latency.
- Use Azure Machine Learning or Azure Cognitive Services.

Benefits

- **Technology choices.** You can mix and match Azure managed services and Apache technologies in HDInsight clusters, to capitalize on existing skills or technology investments.
- **Performance through parallelism.** Big data solutions take advantage of parallelism, enabling high-performance solutions that scale to large volumes of data.
- **Elastic scale.** All of the components in the big data architecture support scale-out provisioning, so that you can adjust your solution to small or large workloads, and pay only for the resources that you use.
- **Interoperability with existing solutions.** The components of the big data architecture are also used for IoT processing and enterprise BI solutions, enabling you to create an integrated solution across data workloads.

Challenges

- **Complexity.** Big data solutions can be extremely complex, with numerous components to handle data ingestion from multiple data sources. It can be challenging to build, test, and troubleshoot big data processes. Moreover, there may be a large number of configuration settings across multiple systems that must be used in order to optimize performance.
- **Skillset.** Many big data technologies are highly specialized, and use frameworks and languages that are not typical of more general application architectures. On the other hand, big data technologies are evolving new APIs that build on more established languages. For example, the U-SQL language in Azure Data Lake Analytics is based on a combination of Transact-SQL and C#. Similarly, SQL-based APIs are available for Hive, HBase, and Spark.
- **Technology maturity.** Many of the technologies used in big data are evolving. While core Hadoop technologies such as Hive and Pig have stabilized, emerging technologies such as Spark introduce extensive changes and enhancements with each new release. Managed services such as Azure Data Lake Analytics and Azure Data Factory are relatively young, compared with other Azure services, and will likely evolve over time.
- **Security.** Big data solutions usually rely on storing all static data in a centralized

data lake. Securing access to this data can be challenging, especially when the data must be ingested and consumed by multiple applications and platforms.

Best practices

- **Leverage parallelism.** Most big data processing technologies distribute the workload across multiple processing units. This requires that static data files are created and stored in a splittable format. Distributed file systems such as HDFS can optimize read and write performance, and the actual processing is performed by multiple cluster nodes in parallel, which reduces overall job times.
- **Partition data.** Batch processing usually happens on a recurring schedule — for example, weekly or monthly. Partition data files, and data structures such as tables, based on temporal periods that match the processing schedule. That simplifies data ingestion and job scheduling, and makes it easier to troubleshoot failures.

Also, partitioning tables that are used in Hive, U-SQL, or SQL queries can significantly improve query performance.

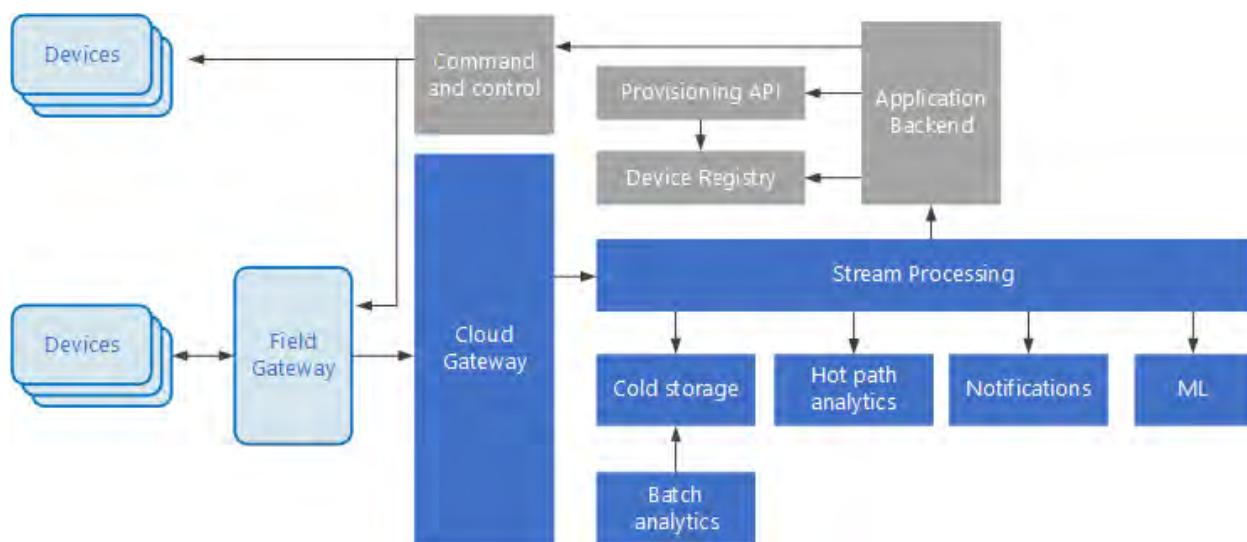
- **Apply schema-on-read semantics.** Using a data lake lets you combine storage for files in multiple formats, whether structured, semi-structured, or unstructured. Use *schema-on-read* semantics, which project a schema onto the data when the data is processing, not when the data is stored. This builds flexibility into the solution, and prevents bottlenecks during data ingestion caused by data validation and type checking.
- **Process data in-place.** Traditional BI solutions often use an extract, transform, and load (ETL) process to move data into a data warehouse. With larger volumes data, and a greater variety of formats, big data solutions generally use variations of ETL, such as transform, extract, and load (TEL). With this approach, the data is processed within the distributed data store, transforming it to the required structure, before moving the transformed data into an analytical data store.
- **Balance utilization and time costs.** For batch processing jobs, it's important to consider two factors: The per-unit cost of the compute nodes, and the per-minute cost of using those nodes to complete the job. For example, a batch job may take eight hours with four cluster nodes. However, it might turn out that the job uses all four nodes only during the first two hours, and after that, only two nodes are required. In that case, running the entire job on two nodes would increase the total job time, but would not double it, so the total cost would be less. In some business scenarios, a longer processing time may be preferable to the higher cost of using underutilized cluster resources.

or using underutilized cluster resources.

- **Separate cluster resources.** When deploying HDInsight clusters, you will normally achieve better performance by provisioning separate cluster resources for each type of workload. For example, although Spark clusters include Hive, if you need to perform extensive processing with both Hive and Spark, you should consider deploying separate dedicated Spark and Hadoop clusters. Similarly, if you are using HBase and Storm for low latency stream processing and Hive for batch processing, consider separate clusters for Storm, HBase, and Hadoop.
- **Orchestrate data ingestion.** In some cases, existing business applications may write data files for batch processing directly into Azure storage blob containers, where they can be consumed by HDInsight or Azure Data Lake Analytics. However, you will often need to orchestrate the ingestion of data from on-premises or external data sources into the data lake. Use an orchestration workflow or pipeline, such as those supported by Azure Data Factory or Oozie, to achieve this in a predictable and centrally manageable fashion.
- **Scrub sensitive data early.** The data ingestion workflow should scrub sensitive data early in the process, to avoid storing it in the data lake.

IoT architecture

Internet of Things (IoT) is a specialized subset of big data solutions. The following diagram shows a possible logical architecture for IoT. The diagram emphasizes the event-streaming components of the architecture.



The **cloud gateway** ingests device events at the cloud boundary, using a reliable, low latency messaging system.

Devices might send events directly to the cloud gateway, or through a **field gateway**. A field gateway is a specialized device or software, usually colocated with the devices, that

Field gateway is a specialized device or software, usually co-located with the devices, that receives events and forwards them to the cloud gateway. The field gateway might also preprocess the raw device events, performing functions such as filtering, aggregation, or protocol transformation.

After ingestion, events go through one or more **stream processors** that can route the data (for example, to storage) or perform analytics and other processing.

The following are some common types of processing. (This list is certainly not exhaustive.)

- Writing event data to cold storage, for archiving or batch analytics.
- Hot path analytics, analyzing the event stream in (near) real time, to detect anomalies, recognize patterns over rolling time windows, or trigger alerts when a specific condition occurs in the stream.
- Handling special types of non-telemetry messages from devices, such as notifications and alarms.
- Machine learning.

The boxes that are shaded gray show components of an IoT system that are not directly related to event streaming, but are included here for completeness.

- The **device registry** is a database of the provisioned devices, including the device IDs and usually device metadata, such as location.
- The **provisioning API** is a common external interface for provisioning and registering new devices.
- Some IoT solutions allow **command and control messages** to be sent to devices.

This section has presented a very high-level view of IoT, and there are many subtleties and challenges to consider. For a more detailed reference architecture and discussion, see the [Microsoft Azure IoT Reference Architecture](#) (PDF download).

Next steps

- Learn more about [big data architectures](#).
- Learn more about [IoT solutions](#).



2003–2023: A Brief History of Big Data

Summing up 20 years of history of Hadoop and everything related



Furcy Pin · [Follow](#)

Published in Towards Data Science

18 min read · Nov 9, 2022

Listen

Share



X



Medium is your home for personal stories, valuable wisdom, and deep expertise.

Become a member and access the best stories on Medium.

[Upgrade for less than \\$5/month](#)

[Or sign up for free](#)

Since its birth and open-sourcing, Hadoop has become the weapon of choice to store and manipulate petabytes of data. A wide and vibrant ecosystem with hundreds of projects has formed around it, and it is still used at many large companies, even if several other cloud-based proprietary solutions are now rivaling it. With this article, I aim to rapidly retrace these 15 years¹ of evolution of the Hadoop ecosystem, explain how it has grown and matured over the past decade, and how the Big Data ecosystem kept evolving in the past few years.

So buckle up for a 20-year travel through time, as our story starts in 2003, in a small town south of San Francisco...

Disclaimer: my initial plan was to illustrate this article with logos of companies and software mentionned, but the extensive use of logos being prohibited on TDS, I decided to keep things entertaining with random images and useless trivia. It's fun to try to remember where we were and what we did at the time.

2003–2006: The Beginning



Medium is your home for personal stories, valuable wisdom, and deep expertise.

Become a member and access the best stories on Medium.

[Upgrade for less than \\$5/month](#)

[Or sign up for free](#)

It all started at the beginning of the millenium, when an already-not-so-small startup in Mountain View called **Google** was trying to index the entirety of the already-not-so-small internet. They had to face two main challenges, yet unsolved at such scale:

How to store hundreds of terabytes of data, on thousands of disks, across more than a thousand machines, with no downtime, data loss, or even data unavailability ?

How to parallelize computation in an efficient and resilient way to handle all this data across all these machines ?

To better understand why this was a difficult problem, consider that when you have a cluster with a thousand machines, there is always *at least* one machine down on average².

From 2003 to 2006, Google released three research papers explaining their internal data architecture, which would change forever the Big Data industry. The first paper came out in 2003 and was entitled “[The Google File System](#)”. The second paper came out in 2004 and was entitled “[MapReduce: Simplified Data Processing on Large Clusters](#)”, and has been cited more than 21 000 times since then, according to Google Scholar. The third one came out in 2006 and was entitled “[Bigtable: A Distributed Storage System for Structured Data](#)”. Even if these papers were essential to the birth of Hadoop, Google did not participate in the birth itself, as they kept their source code proprietary. The story behind that story however is extremely



Medium is your home for personal stories, valuable wisdom, and deep expertise.

Become a member and access the best stories on Medium.

[Upgrade for less than \\$5/month](#)

[Or sign up for free](#)

framework. Unlike Google, Yahoo! decided to open source the project as part of the Apache Software Foundation, thus inviting all the other major tech companies to use and contribute to the project, and help them narrow the technological gap with their neighbors (Yahoo is based in Sunnyvale, next to Mountainview). As we will see, the next few years exceeded the expectations. Of course, Google did quite well too.

2007–2008: Hadoop's early adopters and contributors



**Medium is your home for personal stories,
valuable wisdom, and deep expertise.**

Become a member and access the best stories on Medium.

[Upgrade for less than \\$5/month](#)

[Or sign up for free](#)

capable of converting SQL queries into Map-Reduce jobs on Hadoop, while Cassandra is a wide column store aimed at accessing and updating content in a distributed way on a massive scale. Cassandra did not require Hadoop to function but rapidly became part of the Hadoop ecosystem as connectors for MapReduce were created.

Meanwhile, a lesser known company called **Powerset**, who was working on a search engine, inspired themselves from Google's Bigtable paper to develop *Apache HBase*, another wide column store relying on HDFS for storage. Powerset was soon acquired by **Microsoft**, to bootstrap a new project called *Bing*.

Last but not least, another company had a decisive role in Hadoop's quick adoption: **Amazon**. By starting *Amazon Web Services*, the first on-demand Cloud, and quickly adding support for MapReduce via the *Elastic MapReduce* service, Amazon allowed startups to easily store their data on s3, Amazon's distributed file system, and deploy and run MapReduce jobs on it, without the hassle of managing a Hadoop cluster.

2008–2012: Rise of the Hadoop vendors



Medium is your home for personal stories, valuable wisdom, and deep expertise.

Become a member and access the best stories on Medium.

Upgrade for less than \$5/month

[Or sign up for free](#)

The main pain point of using Hadoop was the great amount of effort required to setup, monitor and maintain a Hadoop cluster. Soon enough the first Hadoop vendor **Cloudera** was founded in 2008, quickly joined by Hadoop's father Doug Cutting. Cloudera proposed a pre-packaged distribution of Hadoop, called **CDH**, along with a cluster monitoring interface **Cloudera Manager**, that finally made it easy to install and maintain a Hadoop cluster, along with its companion softwares like Hive and HBase. **Hortonworks** and **MapR** were founded soon afterwards for the same purpose. Cassandra also got its vendor when **Datastax** was founded in 2010.

Soon enough, everyone agreed that although Hive was a great SQL tool to handle huge ETL batches, it was a poor fit for interactive analytics and BI. Anyone used to standard SQL databases expects them to be able to scan a table with a thousand rows in less than a few milliseconds, where Hive was taking minutes (that's what you get when you ask an elephant to do a mouse's job). This is when a new SQL war started, a war which is still raging today (although we'll see that others have entered the arena since then). Once again, Google had indirectly an enormous influence on the Big Data world, by releasing in 2010 a fourth research paper, called "[Dremel: Interactive Analysis of Web-Scale Datasets](#)". This paper described two major innovations: a distributed interactive query architecture that would inspire most of the interactive SQL that we will mention below, and a column-oriented storage format that would inspire several new data storage format, such as [Apache Parquet](#), developed jointly by Cloudera and Twitter, and [Apache ORC](#), developed jointly by Hortonworks and Facebook.



Medium is your home for personal stories,
valuable wisdom, and deep expertise.

Become a member and access the best stories on Medium.

[Upgrade for less than \\$5/month](#)

[Or sign up for free](#)



Started in 2012: UHDTV, Pinterest, Facebook reaches 1 billion active users, Gagnam Style video reaches 1 billion views on Youtube. Started in 2013: Edward Snowden leaks NSA files, React, Chromecast, Google Glass, Telegram, Slack. (Photo by [Lisa Yount](#) on [Unsplash](#))

While Hadoop was consolidating and adding a new key component, YARN (Yet Another Resource Manager) as its official resource manager, a role that was previously done clumsily by MapReduce, a small revolution began when the open source project Apache Spark started to gain traction at an unprecedented rate. It



Medium is your home for personal stories, valuable wisdom, and deep expertise.

Become a member and access the best stories on Medium.

[Upgrade for less than \\$5/month](#)

[Or sign up for free](#)

connectors to many data sources and format (csv, json, parquet, jdbc, avro, etc.). One interesting thing to note is that Databricks adopted a different market strategy from their predecessors: instead of proposing on-premise deployments for Spark (which Cloudera and Hortonworks quickly added to their own platform), Databricks went for a cloud-only platform offer, starting with AWS (which was by far the most popular cloud at the time), followed by Azure and GCP. Nine years later, we can safely say this was a smart move.

Meanwhile, new projects for dealing with real-time events were open-sourced by other rising tech companies, like *Apache Kafka*, a distributed message queue made by **LinkedIn**, and *Apache Storm*³, a distributed real-time compute engine made by **Twitter**. Both were open-sourced in 2011. Also, during this period, Amazon Web Services were becoming as popular and successful as ever: Netflix's incredible growth in 2010, made mostly possible by Amazon's cloud, would alone illustrate that point. Cloud competitors finally began to arise, with *Microsoft Azure* becoming generally available in 2010, and *Google Cloud Platform* (GCP) in 2011.

2014–2016 Reaching the Apex⁴



Medium is your home for personal stories, valuable wisdom, and deep expertise.

Become a member and access the best stories on Medium.

[Upgrade for less than \\$5/month](#)

[Or sign up for free](#)

them became open-source before that time too. The number of projects started becoming confusing, as we reached the point where for every single need, multiple software solutions existed. More high-level projects also started to emerge like, [Apache Apex](#) (now retired) or [Apache Beam](#) (mostly pushed by Google), aiming at providing a unified interface to handle both batch and streaming processing on top of various distributed back-ends like Apache Spark, Apache Flink or Google's DataFlow.

We can also mention that we finally started to see good open-source schedulers arrive on the market, thanks to [Airbnb](#) and [Spotify](#). Scheduler usage is generally tied to the business logic of the enterprise using it, and it is also a pretty natural and straightforward piece of software to write, at least at first. Then you realize that it is a very hard task to keep it simple and easy to use for others. Which is why pretty much every big tech company has written and (sometimes) open-sourced its own: Yahoo!'s [Apache Oozie](#), LinkedIn's [Azkaban](#), Pinterest's [Pinball](#) (now retired), and many more. However, there never was a wide consensus of one of them being a very good choice, and most companies stuck to their own. Fortunately, around 2015, [Airbnb](#) open-sourced [Apache Airflow](#), while [Spotify](#) open-sourced [Luigi](#)⁵, two schedulers that have quickly reached a high adoption across other companies. In particular, Airflow is now available in SaaS mode on [Google Cloud Platform](#) and [Amazon Web Services](#).

On the SQL side, several other distributed data warehouses emerged, that aimed at providing faster interactive query capabilities than Apache Hive. We already talked



Medium is your home for personal stories, valuable wisdom, and deep expertise.

Become a member and access the best stories on Medium.

[Upgrade for less than \\$5/month](#)

[Or sign up for free](#)



Started in 2016: Oculus Rift, Airpods, Tiktok. Started in 2017: Microsoft Teams, Fortnite. Started in 2018: GDPR, Cambridge Analytica scandal, Among Us. Started in 2019: Disney+, Samsung Galaxy Fold, Google Stadia (Photo by [Jan Canty on Unsplash](#))

In the following years, everything kept accelerating and interconnecting. Keeping up with the list of new technologies and companies in the Big Data market became increasingly difficult, so to keep things short I will speak of four trends that, in my eyes, had the most impact on the Big Data ecosystem.



Medium is your home for personal stories, valuable wisdom, and deep expertise.

Become a member and access the best stories on Medium.

[Upgrade for less than \\$5/month](#)

[Or sign up for free](#)

to catch up with docker, as support for launching Docker containers in Hadoop arrived with version 3.0 in 2018.

The third trend, as mentioned earlier, was the rise of fully managed massively parallel SQL data warehouses for analytics. The rise of the “Modern Data Stack” and of dbt which was first open-sourced in 2016 illustrates that point well.

Finally, the fourth trend that impacted Hadoop was the advent of Deep Learning. In the last half of the 2010’s, everyone has heard about Deep Learning and AI :

AlphaGo passed a milestone by beating the world champion Ke Jie at the game of go, like IBM’s Deep Blue did with Kasparov at chess 20 years before. This technological leap, which already accomplished marvels and promises even more, like self-driving cars, is often associated with Big Data, as it requires to crunch huge amounts of information to be able to train itself. However, Hadoop and Machine Learning were two very different worlds, and they had a hard time working together. In fact, Deep Learning drove the need for new approaches to Big Data, and proved that Hadoop wasn’t the right tool for everything.

Long story short: data scientists working on deep learning needed two things that Hadoop wasn’t able to provide at the time. They needed GPUs, which Hadoop cluster nodes usually did not have, and they needed to install the latest version of their deep learning libraries, such as *Tensorflow* or *Keras*, which was difficult to do on a whole cluster, especially when multiple users were asking for different version of the same library. This particular problem was well addressed by Docker, but the



Medium is your home for personal stories,
valuable wisdom, and deep expertise.

Become a member and access the best stories on Medium.

Upgrade for less than \$5/month

[Or sign up for free](#)

investment firm named CD&R acquired Cloudera at a lower stock price than its initial price.

The decline of Hadoop does not mean its death, though, as many large companies are still using it, especially for on-premise deployments, and all the technologies that were built around it keep using it, or at least parts of it. Innovations are still being made, too. For instance, new storage formats were open-sourced, such as Apache Hudi initially developed at Uber in 2016, Apache Iceberg which was started at Netflix in 2017, and Delta Lake which was open-sourced by Databricks in 2019. Interestingly, one of the main goals behind these new file formats was to circumvent a consequence of the first trend I mentioned: Hive and Spark were initially built for HDFS, and some of the performance properties guaranteed by HDFS were lost in the migration to cloud storages like S3, which caused inefficiencies. But I won't go into the details here, as this particular subject would require another full article.

2020–2023 The modern era



Medium is your home for personal stories, valuable wisdom, and deep expertise.

Become a member and access the best stories on Medium.

Upgrade for less than \$5/month

Or sign up for free

Amazon's Elastic Map Reduce (EMR), Google Dataproc/Dataflow or Azure Synapse. And I have also seen many young companies aim straight for the “Modern Data Stack” approach, built around a SQL analytics warehouse such as *BigQuery*, *Databricks-SQL*, *Athena* or *Snowflake*, fed by no-code (or low-code) data ingestion tools, and organised with dbt, which don't seem to need distributed computing tools like Spark at all. Of course, companies that still prefer on-premise deployments are still using Hadoop and other open-source projects like Spark and Presto, but the proportion of data that is moved to the cloud keeps increasing every year, and I see no reason for that to change for now.

As the data industry kept maturing, we also saw more metadata management and catalog tools being built and adopted. In that scope, we can mention *Apache Atlas*, started by Hortonworks in 2015, *Amundsen*, open-source by Lyft in 2019, and *DataHub*, open-sourced by LinkedIn in 2020. Many private technology startups appeared in that segment, too.

We have also seen startups built around new scheduler technologies , like *Prefect*, *Dagster* and *Flyte*, whose open-source repositories were started in 2017, 2018 and 2019 respectively, and that are challenging Airflow's current hegemony.

Finally, the concept of *lakehouse* has started to emerge. A lakehouse is a platform that combines the advantages of a datalake and a data warehouse⁷. This allows Data Scientists and BI users to work within the same data platform, thus making governance, security, and knowledge sharing easier. Databricks were the first to

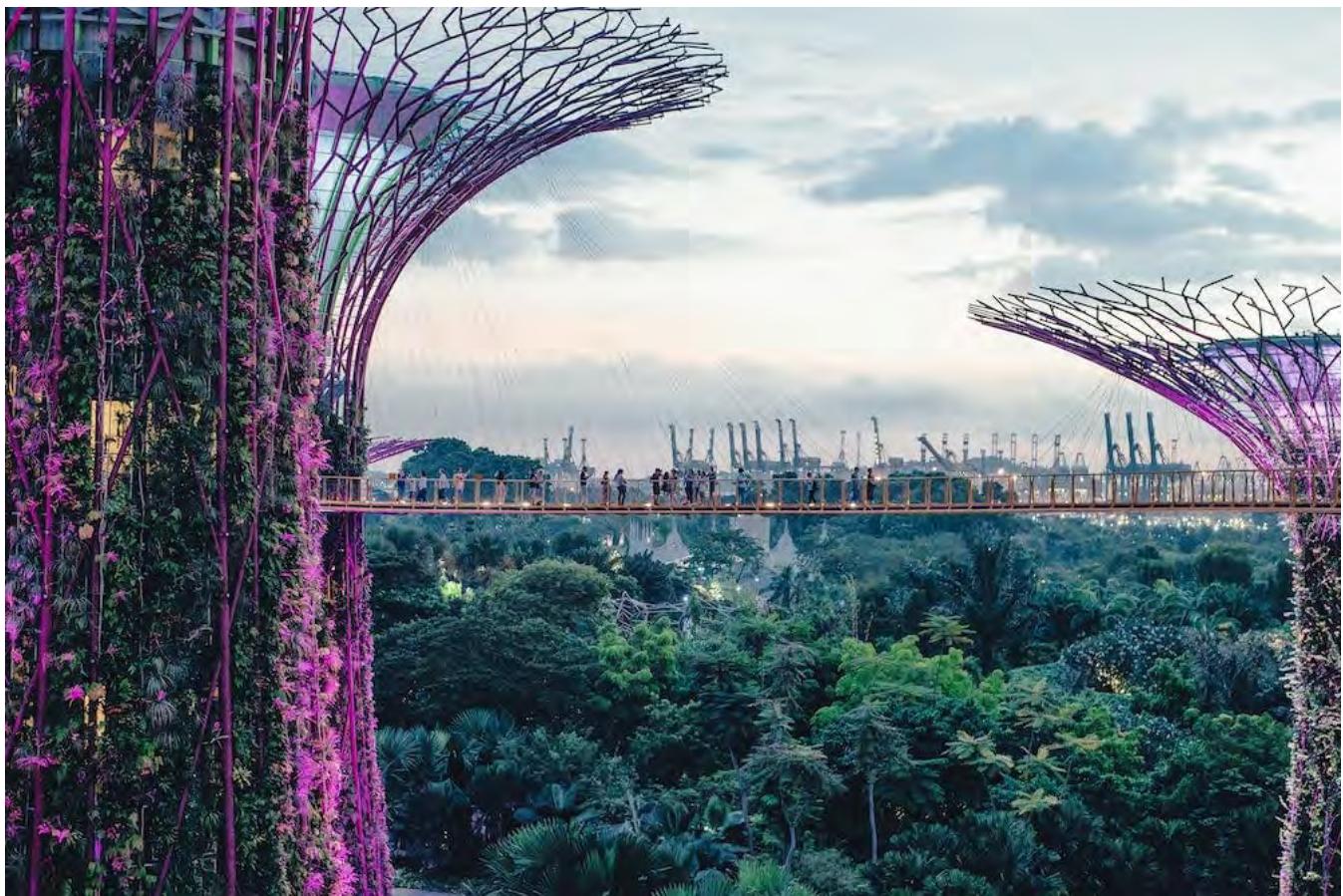


Medium is your home for personal stories, valuable wisdom, and deep expertise.

Become a member and access the best stories on Medium.

Upgrade for less than \$5/month

[Or sign up for free](#)



Started in 2023: who knows ? (Photo by [Annie Spratt](#) on [Unsplash](#))

Since it all started, the number of open-source projects and startups in the Big Data world has kept increasing, year after year (just take a look at the [2021 landscape](#) to see how huge it has become). I remember that around 2012 some people were predicting that the new SQL wars would end and true victors would eventually emerge. This did not happen yet. How all of this will evolve in the future is very



Medium is your home for personal stories, valuable wisdom, and deep expertise.

Become a member and access the best stories on Medium.

[Upgrade for less than \\$5/month](#)

[Or sign up for free](#)

3. Since the beginning, the main lacking resource has been *skilled workforce*. This mean that for most companies⁸, it was simpler to throw more money at performance problems, or migrate to more cost-effective solutions, rather than spend more time optimizing them. Especially now that storage costs in the main distributed warehouses have become so cheap. But perhaps at some point the price competition between vendors will become more difficult to maintain for them, and prices will go up. Even if prices don't go up, the volume of data stored by businesses keeps increasing year after year, and the related cost of inefficiency with them. Perhaps at some point we will see a new trend where people start looking for new, cheaper open-source alternatives, and a new Hadoop-like cycle will start again.

4. In the long term, I believe the real winners will be the cloud providers, Google, Amazon and Microsoft. All they have to do is wait and see in which direction the wind blows the most, bide their time, then acquire (or simply reproduce) the technologies which work the best. Each tool that gets integrated into their cloud makes things so much easier and seamless for users, especially when it comes to security, governance, access control, and cost management. As long as they don't make major organisational mistakes, I don't see how anyone could catch up to them now.

Conclusion

I hope you enjoyed this trip down memory lane with me, and that it helped you better understand (or simply remember) where and how it all started. I tried to



Medium is your home for personal stories, valuable wisdom, and deep expertise.

Become a member and access the best stories on Medium.

[Upgrade for less than \\$5/month](#)

[Or sign up for free](#)

arrival of self-replicating machines, or the arrival of the Raspberry Pi that fueled the DYI movement.

Open source and easy access to knowledge should always be encouraged and fought for, even much more than it is now. It is a never ending battle. One such battle, perhaps the most important one, is happening these days with AI. Large companies did contribute to open-source (for instance Google with TensorFlow), but they also learnt how to use open-source software as venus flytraps to lure users into their proprietary ecosystem, while keeping the most critical (and hardest to replicate) features behind patents.

It is vital for humanity and the world economy that we continue to support open source and knowledge sharing efforts (such as Wikipedia) as best as we can. Governments, citizens, companies and most of all investors must understand this: growth may be driven by innovation, but innovation is driven by sharing knowledge and technologies with the masses.



Medium is your home for personal stories, valuable wisdom, and deep expertise.

Become a member and access the best stories on Medium.

Upgrade for less than \$5/month

Or sign up for free

¹ : It's even 20 years if we count the prequel from Google, hence the title.

² : Maybe in 2022 we made enough progress on hardware reliability to make this less true, but that was definitely the case 20 years ago.

³ : In 2016, [Twitter open sourced Apache Heron](#) (still in the Apache incubation phase, it seems) to replace Apache Storm.

⁴: pun intended.

⁵ : In 2022, [Spotify decided to stop using Luigi and switched to Flyte](#)

⁶: I suspect Apache Beam to be used mostly on GCP with DataFlow.

⁷: [As Databricks puts it](#), a lakehouse combines the flexibility, cost-efficiency, and scale of data lakes with the data management and ACID transactions of data warehouses.

⁸: Of course, I'm not talking about companies the size of Netflix or Uber, here.

Big Data

Hadoop

Apache Spark

Deep Dives



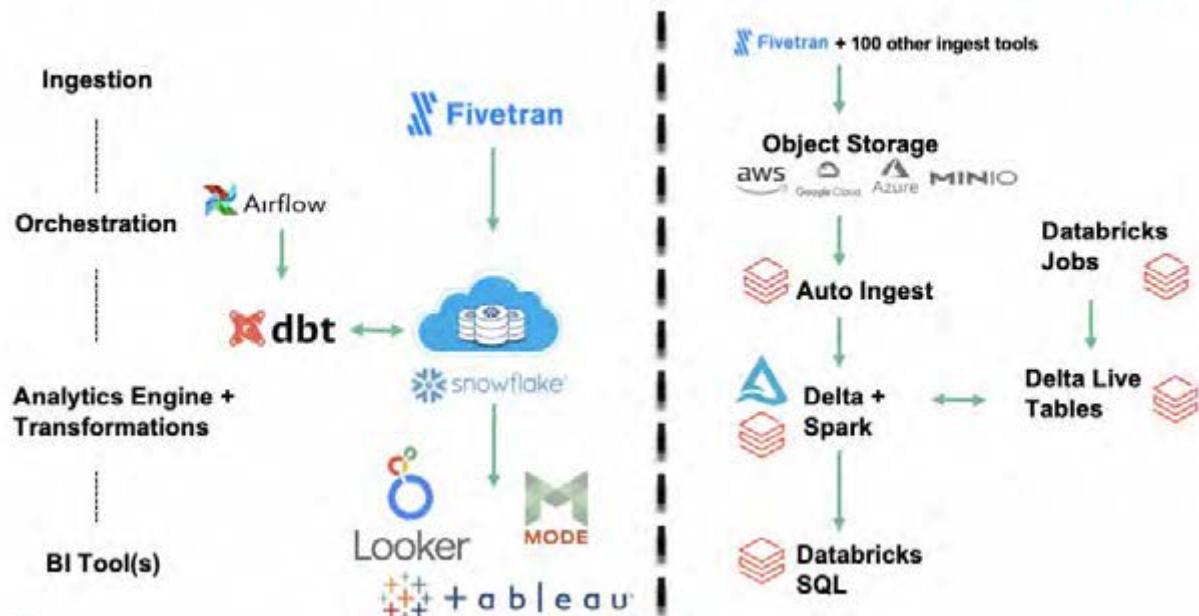
Medium is your home for personal stories, valuable wisdom, and deep expertise.

Become a member and access the best stories on Medium.

Upgrade for less than \$5/month

[Or sign up for free](#)

MODERN DATA STACK VS DATAWAREHOUSE ECOSYSTEM



Furcy Pin in Towards Data Science

Modern Data Stack: which place for Spark ?

One year ago, some were already predicting that dbt will one day become bigger than Spark, and the year 2021 proved them right: dbt has...

7 min read · Jan 26, 2022

687

15



Replace this



Medium is your home for personal stories, valuable wisdom, and deep expertise.

Become a member and access the best stories on Medium.

Upgrade for less than \$5/month

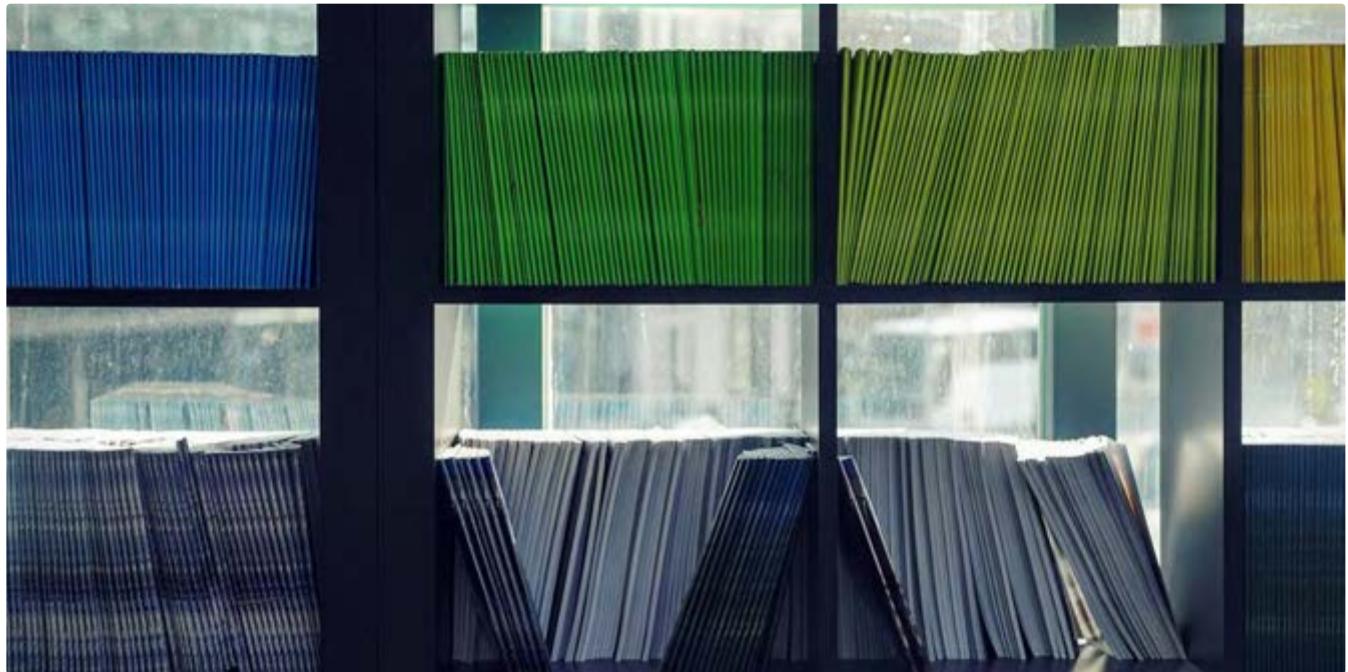
Or sign up for free

And How to Efficiently Interact with Config Files in Python

◆ · 6 min read · May 26

👏 2K

💬 21



Jacob Marks, Ph.D. in Towards Data Science

How I Turned My Company's Docs into a Searchable Database with OpenAI

And how you can do the same with your docs



Medium is your home for personal stories, valuable wisdom, and deep expertise.

Become a member and access the best stories on Medium.

Upgrade for less than \$5/month

[Or sign up for free](#)



Furcy Pin in Towards Data Science

SQL + jinja is not enough—why we need DataFrames

After reading Max Beauchemin words about “Mountains of Templatized SQL and YAML” I decided to start a POC to illustrate that point and show ...

20 min read · Feb 28, 2022



415



See all from Furcy Pin



Medium is your home for personal stories, valuable wisdom, and deep expertise.

Become a member and access the best stories on Medium.

Upgrade for less than \$5/month

[Or sign up for free](#)



 Pier Paolo Ippolito in Towards Data Science

Apache Spark Optimization Techniques

A review of some of the most common Spark performance problems and how to address them

◆ · 5 min read · Jan 11

 101  2





Medium is your home for personal stories, valuable wisdom, and deep expertise.

Become a member and access the best stories on Medium.

Upgrade for less than \$5/month

[Or sign up for free](#)

Metadata, Statistics on Row Groups, Partitions discovery, and Repartitioning

◆ · 8 min read · Jan 3

👏 467

💬 1

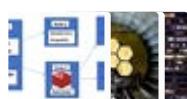


Lists



Stories to Help You Grow as a Software Developer

19 stories · 156 saves



New_Reading_List

173 stories · 8 saves



Stories to Help You Level-Up at Work

19 stories · 129 saves



Natural Language Processing

373 stories · 27 saves

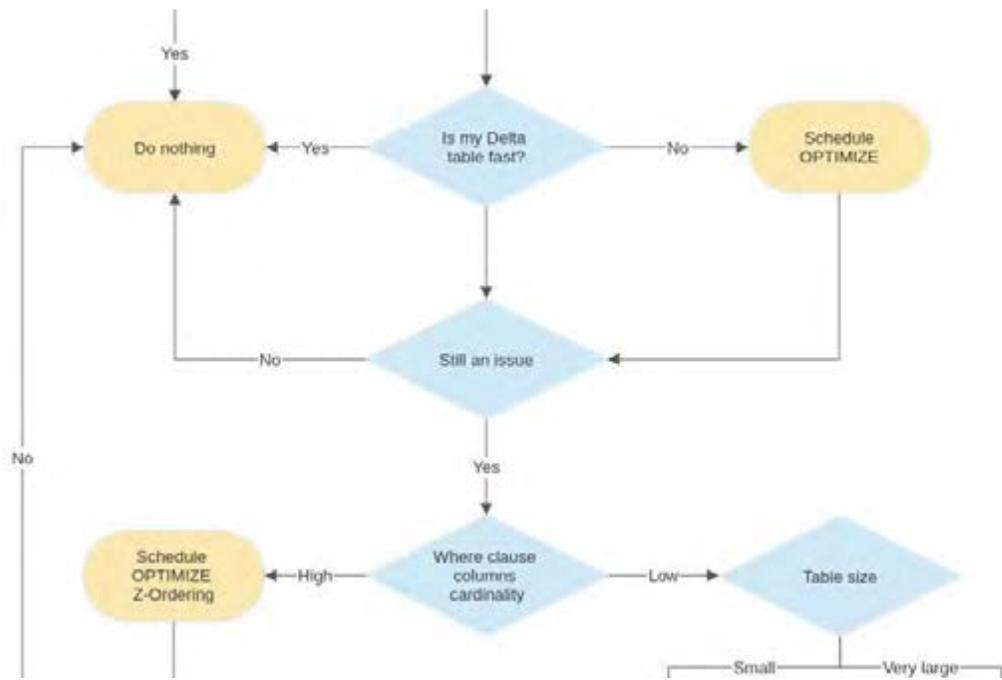


Medium is your home for personal stories, valuable wisdom, and deep expertise.

Become a member and access the best stories on Medium.

Upgrade for less than \$5/month

[Or sign up for free](#)



Vitor Teixeira in Towards Data Science

Delta Lake— Keeping it fast and clean

Ever wondered how to improve your Delta tables' performance? Hands-on on how to keep Delta tables fast and clean.

★ · 11 min read · Feb 15

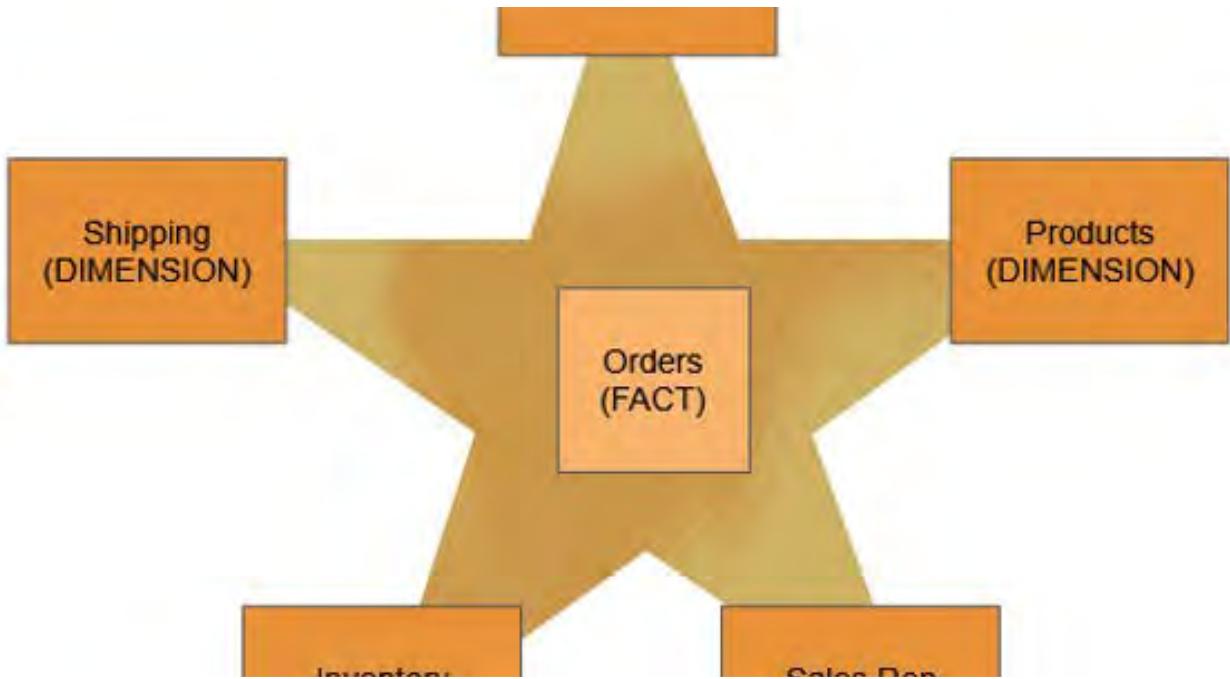


Medium is your home for personal stories, valuable wisdom, and deep expertise.

Become a member and access the best stories on Medium.

[Upgrade for less than \\$5/month](#)

[Or sign up for free](#)



Manoj Kukreja in Towards Data Science

Handling Slowly Changing Dimensions (SCD) using Delta Tables

Handling the challenge of slowly changing dimensions using the Delta Framework

◆ · 10 min read · Jan 23



118



1



Medium is your home for personal stories, valuable wisdom, and deep expertise.

Become a member and access the best stories on Medium.

Upgrade for less than \$5/month

[Or sign up for free](#)

Apache Airflow is an orchestration tool developed by Airbnb and later given to the open-source community. Today, it is the most beloved...

★ · 9 min read · Feb 8

161

1

W+

See more recommendations



Medium is your home for personal stories, valuable wisdom, and deep expertise.

Become a member and access the best stories on Medium.

Upgrade for less than \$5/month

[Or sign up for free](#)

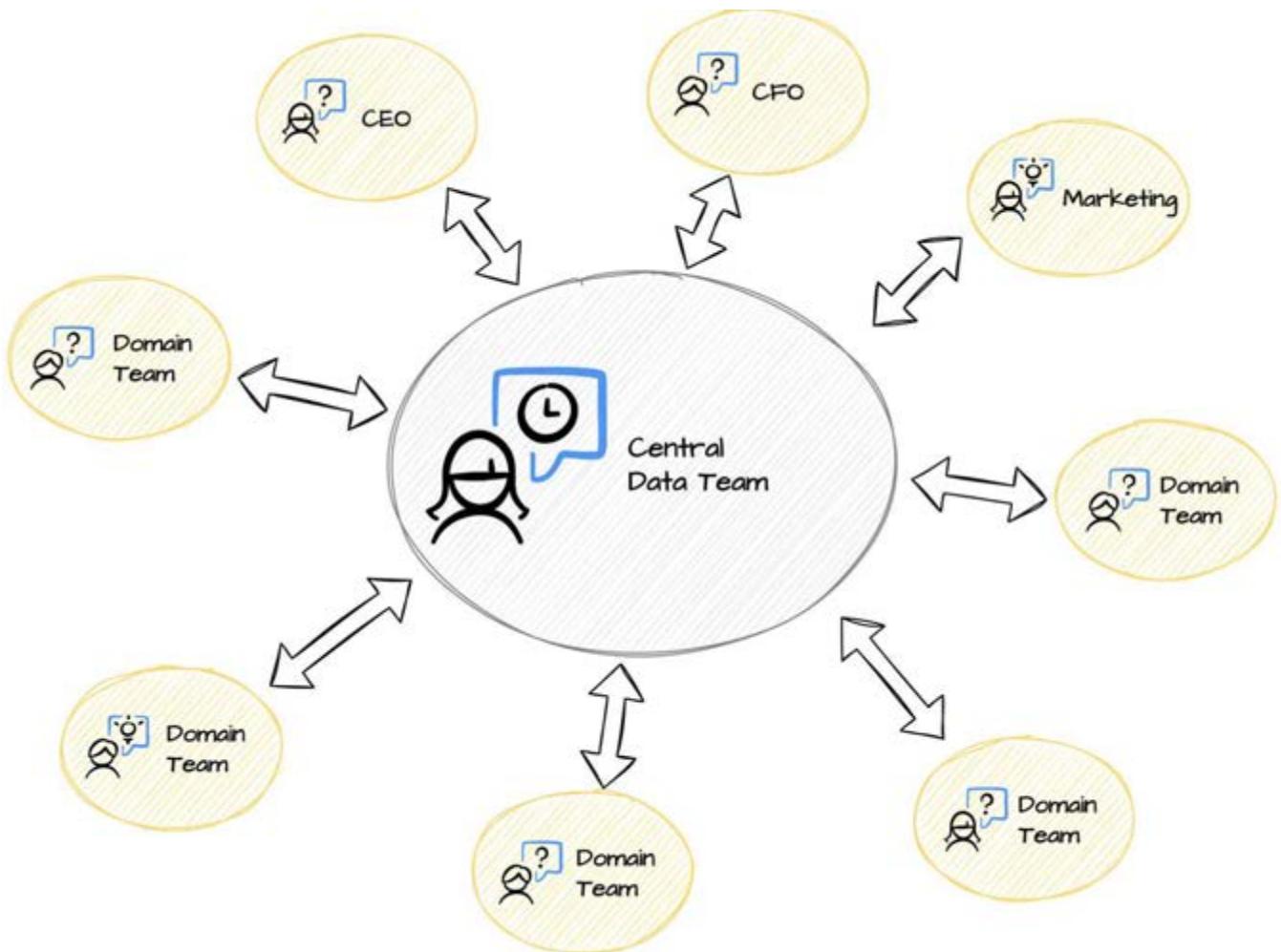
NEW: We launched Data Mesh Manager to build a data product inventory and manage data contracts.

DATA MESH ARCHITECTURE

Data Mesh From an Engineering Perspective



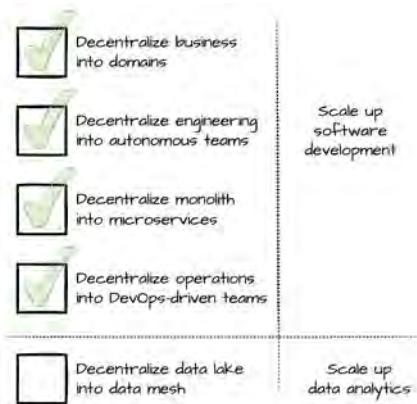
Why You May Need a Data Mesh



Many organizations have invested in a central data lake and a data team with the expectation to drive their business based on data. However, after a few initial quick wins, they notice that **the central data team often becomes a bottleneck**. The team cannot handle all the analytical questions of management and product owners quickly enough. This is a massive problem because making timely data-driven decisions is crucial to stay competitive. For example: Is it a good idea to offer free shipping during Black Week? Do customers accept longer but more reliable shipping times? How does a product page change influence the checkout and returns rate?

The data team wants to answer all those questions quickly. In practice, however, they struggle because they need to spend too much time fixing broken data pipelines after operational database changes. In their little time remaining, **the data team has to discover and understand the necessary domain data**. For every question, they need to learn domain knowledge to give meaningful insights. Getting the required domain expertise is a daunting task.

On the other hand, organizations have also invested in domain-driven design,



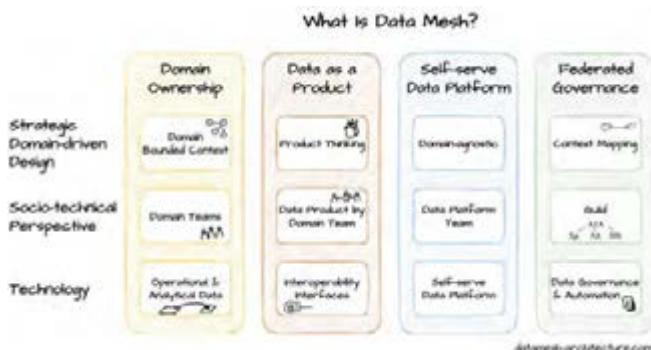
autonomous domain teams (also known as stream-aligned teams or product teams) and a decentralized microservice architecture. These **domain teams own and know their domain**, including the information needs of the business. They design, build, and run their web applications and APIs on their own. Despite knowing the domain and the relevant information needs, the domain teams have to reach out to the overloaded central data team to get the necessary data-driven insights.

With the eventual growth of the organization, the situation of the domain teams and the central data team becomes worse. A way out of this is to shift the responsibility for data from the central data team to the domain teams. This is the core idea behind the data mesh concept: **Domain-oriented decentralization for analytical data**. A data mesh architecture enables domain teams to perform cross-domain data analysis on their own and interconnects data, similar to APIs in a microservice architecture.

What Is Data Mesh?

The term *data mesh* was coined by [Zhamak Dehghani](#) in 2019 and is based on four fundamental principles that bundle well-known concepts:

The **domain ownership** principle mandates the domain teams to take responsibility for their data. According to this principle, analytical data should be composed around domains, similar to the team boundaries aligning with the system's bounded context. Following the domain-driven distributed



architecture, analytical and operational data ownership is moved to the domain teams, away from the central data team.

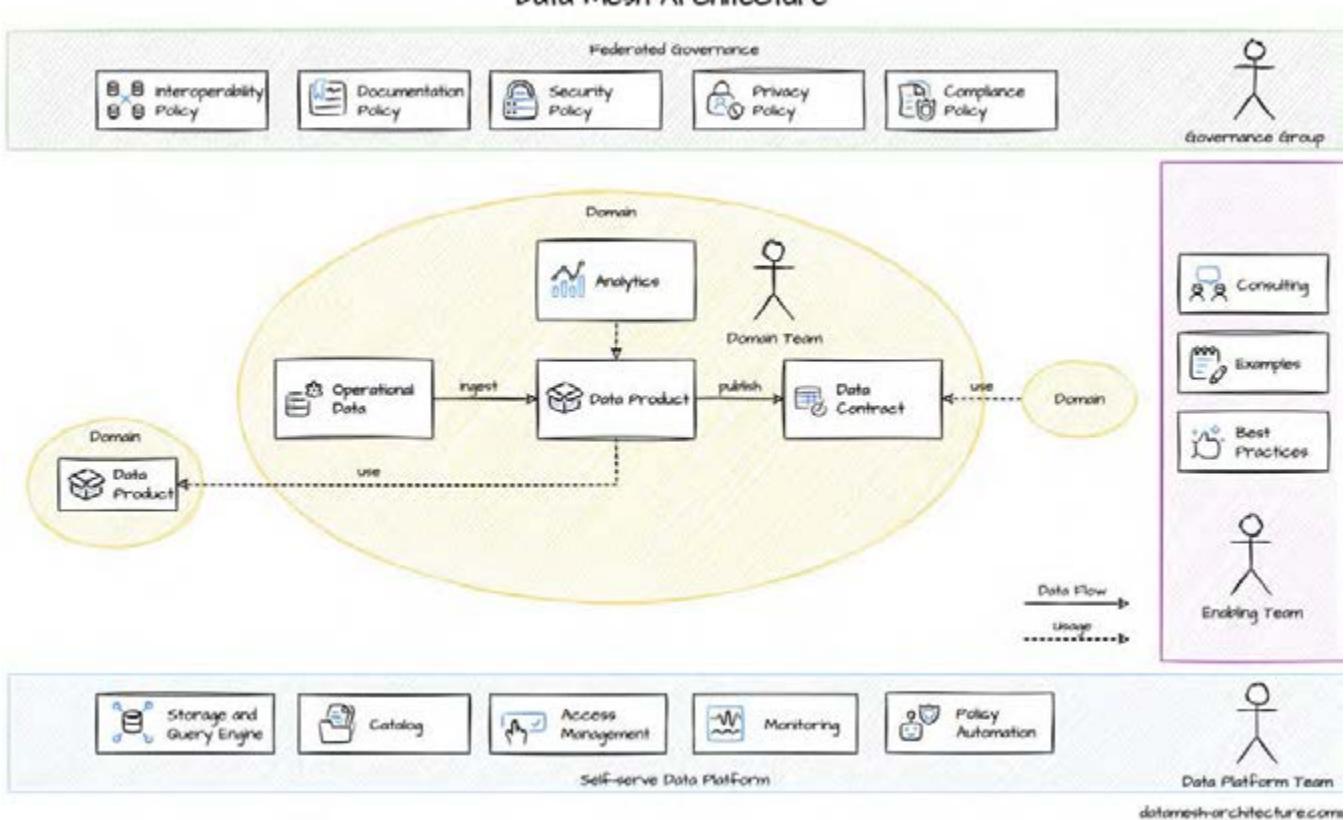
The **data as a product** principle projects a product thinking philosophy onto analytical data. This principle means that there are consumers for the data beyond the domain. The domain team is responsible for satisfying the needs of other domains by providing high-quality data. Basically, domain data should be treated as any other public API.

The idea behind the **self-serve data infrastructure platform** is to adopt platform thinking to data infrastructure. A dedicated data platform team provides domain-agnostic functionality, tools, and systems to build, execute, and maintain interoperable data products for all domains. With its platform, the data platform team enables domain teams to seamlessly consume and create data products.

The **federated governance** principle achieves interoperability of all data products through standardization, which is promoted through the whole data mesh by the governance group. The main goal of federated governance is to create a data ecosystem with adherence to the organizational rules and industry regulations.

How To Design a Data Mesh?

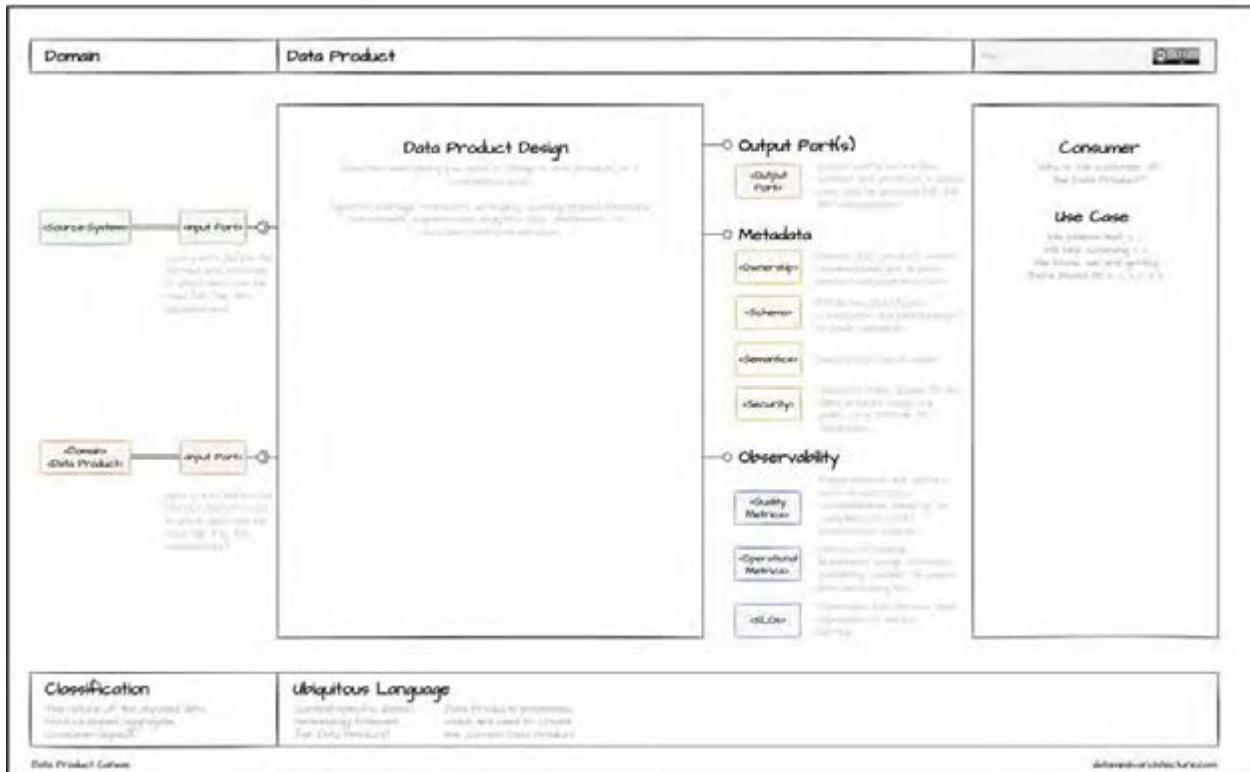
A data mesh architecture is a decentralized approach that enables domain teams to perform cross-domain data analysis on their own. At its core is the domain with its responsible team and its operational and analytical data. The domain team ingests operational data and builds analytical data models as data products to perform their own analysis. It may also choose to publish data products with data contracts to serve other domains' data needs.



The domain team agrees with others on global policies, such as interoperability, security, and documentation standards in a federated governance group, so that domain teams know how to discover, understand and use data products available in the data mesh. The self-serve domain-agnostic data platform, provided by the data platform team, enables domain teams to easily build their own data products and do their own analysis effectively. An enabling team guides domain teams on how to model analytical data, use the data platform, and build and maintain interoperable data products.

Let's zoom in to the core components of a data mesh architecture and their relationships:

Data Product



A data product is a logical unit that contains all components to process and store domain data for analytical or data-intensive use cases and makes them available to other teams via output ports. You can think of a microservice, but for analytical data.

Data products connect to sources, such as operational systems or other data products and perform data transformation. Data products serve data sets in one or many output ports. Some examples:

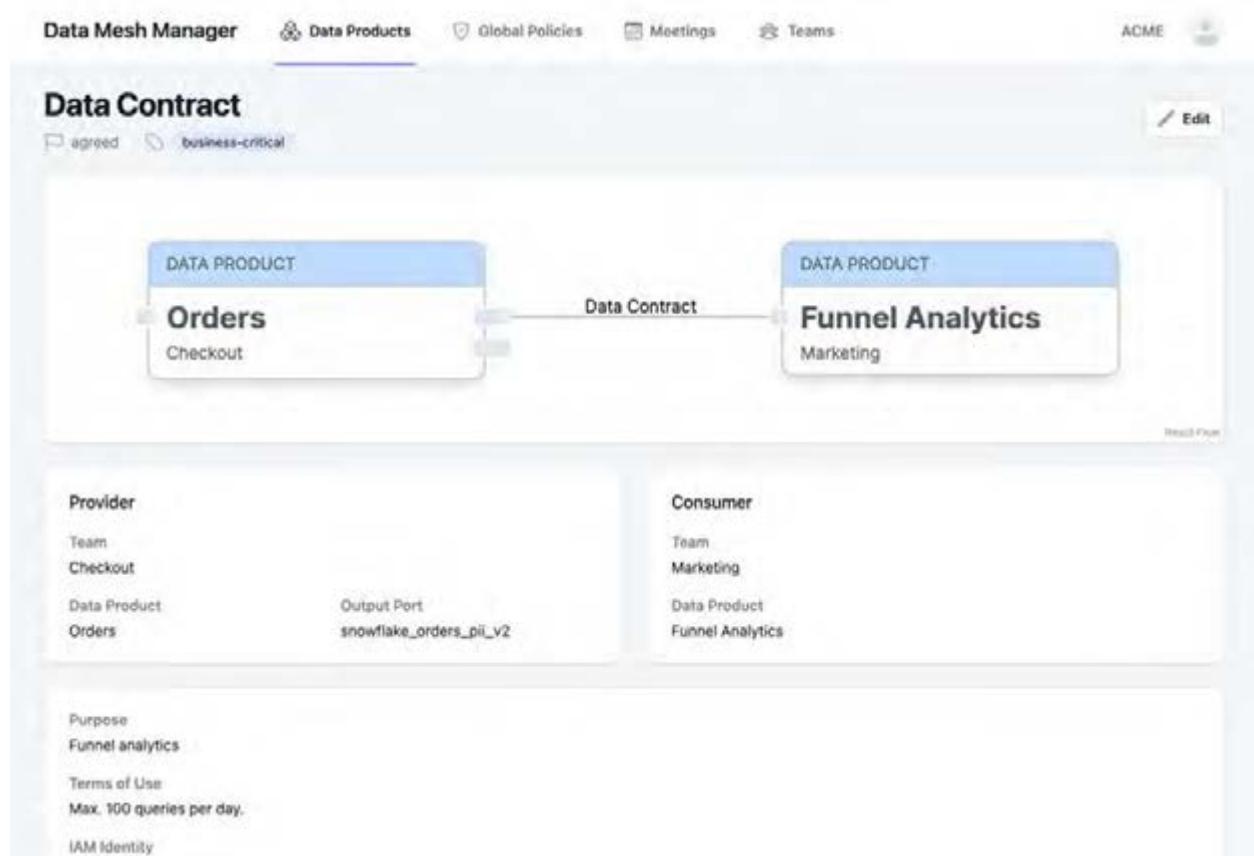
- A BigQuery dataset with multiple related tables
- Parquet files in an AWS S3 bucket
- Delta files in Azure Data Lake Storage Gen2
- Dashboards in Looker
- A machine learning model as an ONNX file

When data products provide data for other teams, a data contract defines the endpoint, syntax, semantics, and quality of provided data, similar to OpenAPI or AsyncAPI specifications. Data contracts are usually defined in a central data catalog, along with further metadata. Currently, there is no industry standard for data contracts.

A data product is owned by a domain team. The team is responsible for the operations of the data product during its entire lifecycle. The team needs to continuously monitor and ensure data quality, availability, and costs. For example, keep the data without duplicates or react to missing entries.

To design data products, we recommend to use the **Data Product Canvas**.

Data Contract



A data contract is a formal agreement between two parties to use a data product. It specifies the guarantees about a provided data set and expectations concerning data product usage. It covers:

- Data Product Provider, including team, owner, and the output port to access
- Data Product Consumer, including team, and responsible contact
- Purpose of data usage
- Schema and semantics of used data attributes
- Service-level objectives, such as latency, availability, availability
- Terms, such as query intervals and data processing volumes
- Costs
- Start date
- End date
- Notice period, to cancel the contract

Data contracts are the links between data products. They typically refer to a specific version of an output port. They become active when a consumer requests data access and the provider accepts the request. Data contracts can be canceled by providers or consumers with a defined notice period, e.g., when the costs of usage become too high, or when the provider needs to implement a breaking change. Data contracts provide

stability, trust, and quality within the data mesh. They can be used to visualize data flow across data products.

Data contracts can also be used for automation: As soon as a data contract has been concluded, permissions for the output port can be set up automatically in the data platform on an event basis. When the contract is terminated, the permissions are automatically deleted again.

As there is a lack of tooling support, we built **Data Mesh Manager** to manage data contracts.

Federated Governance

The screenshot shows the Data Mesh Manager application interface. At the top, there are navigation tabs: Data Mesh Manager, Data Products, Global Policies (which is the active tab), Meetings, and Teams. Below the tabs, the breadcrumb navigation shows: Home > Global Policies > Parquet File Format. The main content area is titled "Parquet File Format". It includes a status bar with "GLOBAL-3", "Interoperability", and "Accepted". On the right, there is an "Edit" button. The page is divided into sections: "Context" (describing the need for a common file format), "Decision" (stating Apache Parquet is used), "Consequences" (listing benefits like low storage costs and fast processing), and "Adoption" (listing teams that have adopted the policy). The "Adoption" section includes checkboxes for "Customers", "Orders", and "Payment Fraud Report", all of which are checked. Other sections like "Controlling" and "Marketing" also have some items checked.

The federated governance group is typically organized as a guild consisting of representatives of all teams taking part in the data mesh. They agree on global policies, which are the rules of play in the data mesh. These rules define how the domain teams have to build their data products.

Policies on **interoperability** are the starting point. They allow other domain teams to use data products in a consistent way. For example, global policies could define that the

standard way to provide data is as a CSV file on AWS S3 in a bucket owned by the corresponding domain team.

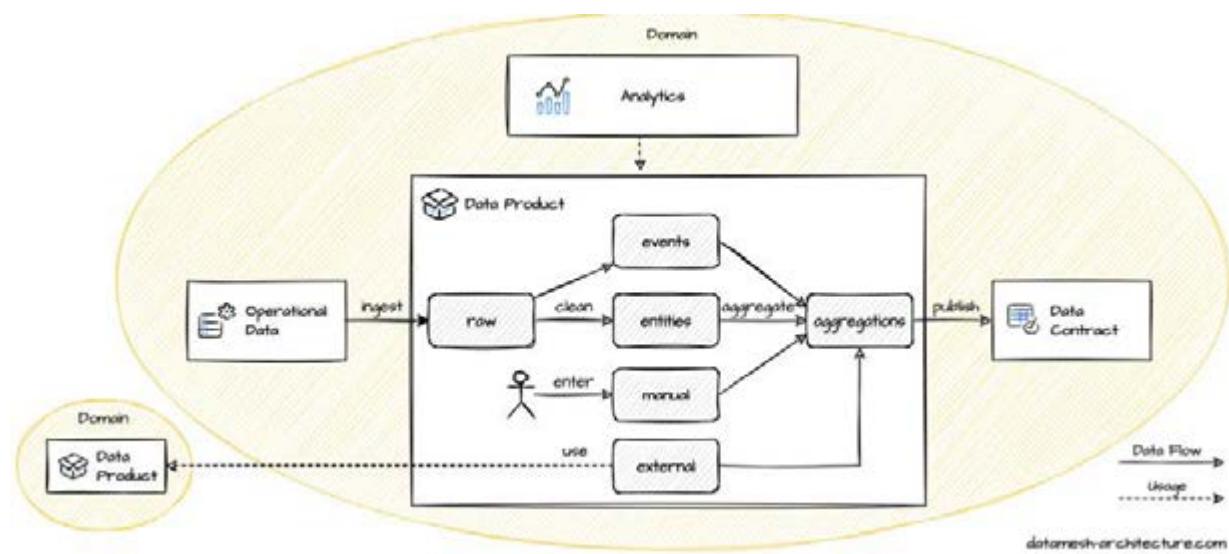
Next, there has to be some form of **documentation** to discover and understand available data products. A simple policy for this could be a wiki page with a predefined set of metadata, such as owner of the data product, location URL, and descriptions of the CSV fields.

A uniform way to access the actual data product in a **secure** way could be using role-based access in AWS IAM, managed by the domain team.

Global policies such as **privacy** and **compliance** are also common. Think about protection of personally identifiable information (PII) or industry-specific legal requirements.

Lots of example policies are available on our other website datamesh-governance.com that you easily use in the **Data Mesh Manager**, our tool for data mesh governance.

Transformations



Diving into the organization of data within a data product, we can see the different kind of data that flows through different stages. Operational data is often ingested as some kind of **raw** and unstructured data.

In a preprocessing step, raw data is cleaned and structured into events and entities.

Events are small, immutable, and highly domain oriented, such as *OrderPurchased* or *ShipmentDelivered*. **Entities** represent business objects such as *shipments* or *articles* with their state changing over time. That's why the entities often are represented as a list of snapshots, the history, with the latest snapshot being the current state.

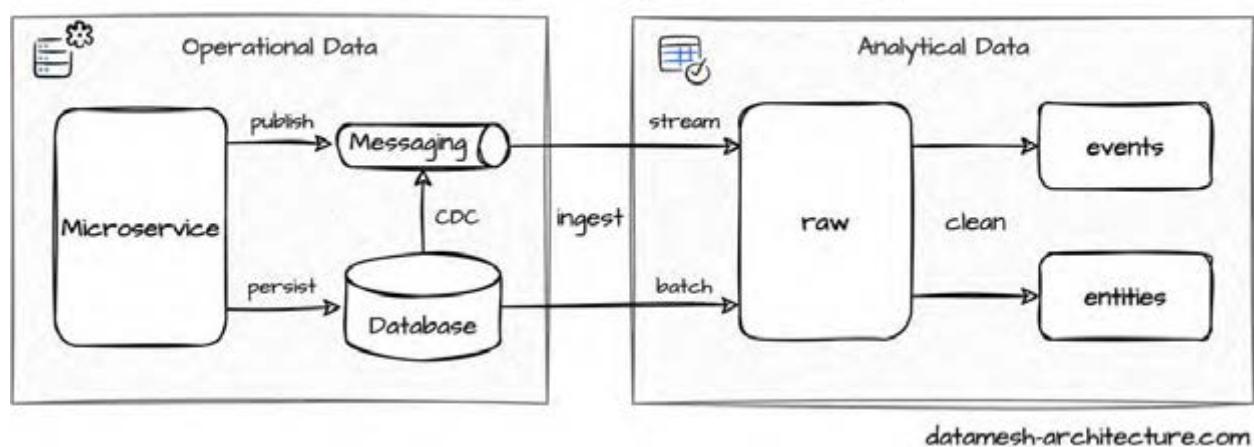
In practice, we often see **manually** entered or imported data. For example, forecast data sent via email as CSV files or text descriptions for business codes.

Data from other teams are integrated as **external** data. When using data products from other teams that are well-governed, this integration might be implemented in a very lightweight way. In case of importing data from legacy systems, the external area acts as an anti-corruption layer ↗.

Aggregations combine data to answer analytical questions. Domain data can be published to other teams by defining a data contract. The data contract is usually implemented by a view, that is stable, even when the underlying data models change.

Ingesting

Data Ingesting and Cleaning



How can domain teams ingest their operational data into the data platform? A software system designed according to domain-driven design principles contains data as mutable entities/aggregates and immutable domain events.

Domain events are a great fit to be ingested into the data platform as they represent relevant business facts. If there's a messaging system in place domain events can be forwarded to the data platform by attaching an additional message consumer. Data can be collected, processed, and forwarded to the data platform in real time. With this **streaming ingestion**, data is sent in small batches when they arrive, so they are immediately available for analytics. As domain events are already well defined, there is little to do in terms of cleaning and preprocessing, except deduplication and anonymization of PII data. Sometimes, it is also advisable to define and ingest internal analytical events that contain information that is relevant only for analytical use cases so that domain events don't have to be modified.

Examples for streaming ingestion: Kafka Connect, Kafka Streams, AWS Lambda

Many business objects are persisted as **entities and aggregates** in SQL or NoSQL databases. Their state changes over time, and the latest state is persisted in the database only. Strong candidates for entities with state are *articles*, *prices*, *customer data*, or *shipment status*. For analytical use cases, it is often required to have both the latest state and the history of states over time. There are several approaches to ingest entities. One way is to generate and publish an **onCreate/onUpdateonDelete event** with the current state every time an entity is changed, e.g. by adding an aspect [↗](#) or EntityListeners [↗](#). Then streaming ingestion can be used to ingest the data as described above. When it is not feasible to change the operational software, **change data capture (CDC)** may be used to listen to database changes directly and stream them into the data platform.

Examples for CDC streaming: [Debezium ↗](#)

Lastly, traditional scheduled **ELT or ETL jobs** that export data to file and load them into the platform can be set up, with the downside of not having real-time data, not having all stage changes between exports, and some work to consolidate exported data again. However, they are a viable option for legacy systems, such as mainframes.

Clean Data

```
1  -- Step 1: Deduplicate
2  WITH inventory_deduplicated AS (
3      SELECT *
4      EXCEPT (row_number)
5      FROM (
6          SELECT *,
7              ROW_NUMBER() OVER (PARTITION BY
8              FROM `datameshexample-fulfillment`
9              WHERE row_number = 1
10     ),
11  -- Step 2: Parse JSON to columns
12  inventory_parsed AS (
13      SELECT
14          json_value(data,("$.sku"))
15          json_value(data,("$.location"))
16          CAST(json_value(data,("$.available")) AS
17          CAST(json_value(data,("$.updated")) AS
18      FROM inventory_deduplicated
19  )
20  -- Step 3: Actual Query
21  SELECT sku, location, available, updated
22  FROM inventory_parsed
```

```
23 ORDER BY sku, location, updated_at
```

entities_inventory_history.sql hosted with [view raw](#)
 by GitHub

Clean data is the foundation for effective data analytics. With data mesh, domain teams are responsible for performing data cleaning. They know their domain and can identify why and how their domain data needs to be processed.

Data that is ingested into the data platform is usually imported in its original raw and unstructured format. When using a columnar database, this might be a row per event that contains a **CLOB** ↗ field for the event payload, which may be in JSON format. Now it can be preprocessed to get data clean:

- **Structuring:** Transform unstructured and semi-structured data to the analytical data model, e.g., by extracting JSON fields into columns.
- **Mitigation of structural changes:** When data structures have changed, mitigate them, e.g., by filling null values with sensible defaults.
- **Deduplication:** As most analytical storage systems are append-only, entities and events cannot be updated. Remove all duplicate entries.
- **Completeness:** Ensure that data contain agreed periods, even when there were technical issues during ingestion.
- **Fix outliers:** Invalid data that may occur through bugs get identified and corrected.

From an implementation perspective, these preprocessing steps can be implemented as simple SQL views that project the raw data. The queries may be organized through **common table expressions** ↗ (CTEs) and may be enhanced with **user-defined functions** ↗ (UDFs), e.g., for JSON processing. As an alternative, the cleaning steps can be implemented as lambda functions that operate on topics. More complex pipelines can be built with frameworks like **dbt** ↗ or **Apache Beam** ↗ that offer an advanced programming model, but also require more skills to master.

Analytics

The screenshot shows a Jupyter Notebook interface with the title "Data Mesh Example - Inventory.ipynb". The code cell contains Python code for authenticating with Google Colab:

```
[1]: from google.colab import auth  
auth.authenticate_user()  
print('Authenticated')  
Authenticated
```

Another code cell loads the `google.colab.data_table` extension:

```
[2]: %load_ext google.colab.data_table
```

A text cell asks, "How is current inventory distributed over locations?" followed by a BigQuery query:

```
▶ %%bigquery --project datameshexample-fulfillment  
SELECT location, sum(available) as available_total  
FROM `aggregates.inventory_latest`  
group by location  
order by location
```

The output is a data table showing the distribution of inventory across locations:

index	location	available_total
0	10	84
1	11	99
2	12	77
3	13	79
4	14	84
5	15	99
6	16	100
7	17	80
8	18	63
9	19	65

Page navigation controls show "Show 10" and "per page" with a dropdown, and a page number indicator "1 2".

To gain insights, domain teams query, process, and aggregate their analytical data together with relevant data products from other domains.

SQL is the foundation for most analytical queries. It provides powerful functions to connect and investigate data. The data platform should perform join operations efficiently, even for large data sets. Aggregations are used to group data and window functions help to perform a calculation across multiple rows. Notebooks help to build and document exploratory findings.

Examples: Jupyter Notebooks, Presto

Humans understand data, trends, and anomalies much easier when they perceive them visually. There are a number of great data **visualization** tools that build beautiful charts, key performance indicator overviews, dashboards and reports. They provide an easy-to-use UI to drill down, filter, and aggregate data.

Examples: Looker, Tableau, Metabase, Redash

For more advanced insights, **data science and machine learning** methods can be applied. These enable correlation analyses, prediction models, and other advanced use cases. Special methodological, statistical, and technological skills are required.

Examples: *scikit-learn*, *PyTorch*, *TensorFlow*

Data Platform

The self-serve data platform may vary for each organization. Data mesh is a new field and vendors are starting to add data mesh capabilities to their existing offerings.

Looking from the desired capabilities, you can distinguish between analytical capabilities and data product capabilities: **Analytical capabilities** enable the domain team to build an analytical data model and perform analytics for data-driven decisions. The data platform needs functions to ingest, store, query, and visualize data as a self-service. Typical data warehouse and data lake solutions, whether on-premise or a cloud provider, already exist. The major difference is that each domain team gets its own isolated area.

A more advanced data platform for data mesh also provides additional domain-agnostic **data product capabilities** for creating, monitoring, discovering, and accessing data products. The self-serve data platform should support the domain teams so that they can quickly build a data product as well as run it in production in their isolated area. The platform should support the domain team in publishing their data products so that other teams can discover them. The discovery requires a central entry point for all the decentralized data products. A data catalog can be implemented in different ways: as a wiki, git repository, or there are even already vendor solutions for a cloud-based data catalog such as Select Star, Google Data Catalog, or AWS Glue Data Catalog. The actual usage of data products, however, requires a domain team to access, integrate, and query other domains' data products. The platform should support, monitor, and document the cross-domain access and usage of data products.

An even more advanced data platform supports **policy automation**. This means that, instead of forcing the domain team to manually ensure that the global policies are not violated, the policies are automatically enforced through the platform. For example, that all data products have the same metadata structure in the data catalog, or that the PII data are automatically removed during data ingestion.

Efficiently combining data products from multiple domains, i.e., having large cross-domain join operations within a few seconds, ensures developer acceptance and happiness. That's why the **query engine has a large influence on the architecture of the data platform**. A shared platform with a single query language and support for separated areas is a good way to start as everything is highly integrated. This could be Google BigQuery with tables in multiple projects that are discoverable through Google

Data Catalog. In a more decentralized and distributed data mesh, a distributed query engine such as Presto can still perform cross-domain joins without importing data, but they come with their own limitations, e.g., limited pushdowns require that all underlying column data need to be transferred.

Enabling Team

The enabling team spreads the idea of data mesh within the organization. In the beginning of data mesh adoption, a lot of explanatory efforts will be required and the enabling team can act as data mesh advocates. They help domain teams [on their journey to become a full member of the data mesh](#). The enabling team consists of specialists with extensive knowledge on data analytics, data engineering, and the self-serve data platform.

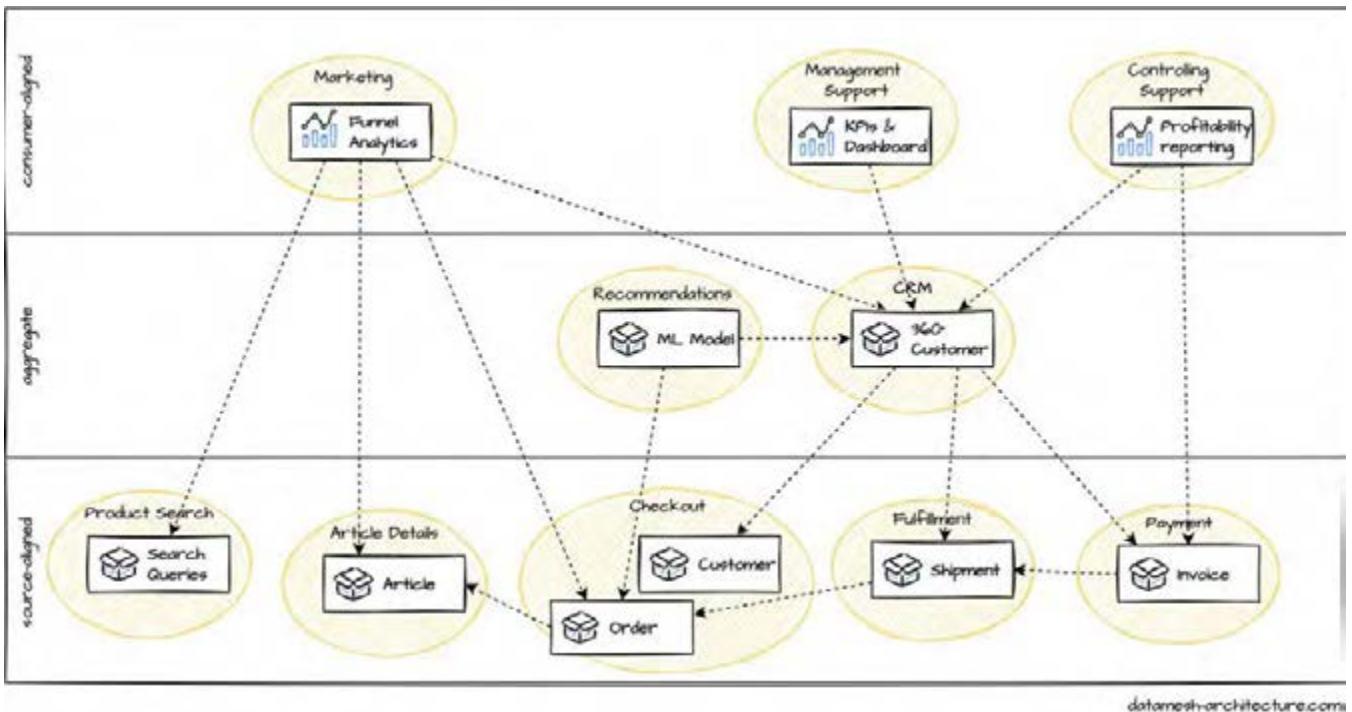
A member of the enabling team temporarily joins a domain team for a limited time span like a month as an **internal consultant** to understand the team's needs, establish a learning environment, upskill the team members in data analytics, and guide them on how to use the self-serve data platform. They don't create data products by themselves.

In between their consulting engagements, they **share learning materials** such as walking skeletons, examples, best practices, tutorials, or even podcasts.

Mesh

The *mesh* emerges when teams use other domain's data products. Using data from upstream domains simplifies data references and lookups (such as getting an article's price), while data from downstream domains enables analyzing effects, e.g. for A/B tests (such as changes in the conversion rate). Data from multiple other domains can be aggregated to build comprehensive reports and new data products.

Let's look at a simplified e-commerce example:



datamest-architecture.com

Domains can be classified by data characteristics and data product usage. We adopt Zhamak Dehghani's classification:

Source-aligned

In this example, an online shop is subdivided into domains along the customer journey, from *product search* over *checkout* to *payment*. In a data mesh, these domains publish their data as data products, so others can access them. The engineers do analytics on their own data to improve their operational systems and validate the business value of new features. They use domain neighbor's data to simplify their queries and get insights on effects in downstream domains. These domain data can be referred to as **source-aligned**, as most of their published data products correspond closely to the *domain events* and *entities* generated in their operational systems.

Aggregate

For **complicated subsystems** ↗, it can be efficient that a team focuses solely on delivering a data product that is **aggregated** of various data products from other domains. A typical example is a 360° customer view that includes relevant data from multiple domains, such as account data, orders, shipments, invoices, returns, account balance, and internal ratings. With respect to different bounded contexts, a comprehensive 360° customer view is hard to build, but it might be useful for many other domains. Another example for a complicated subsystem is building sophisticated ML models that require enhanced data science skills. It may be sensible that a data scientists team develops and trains a recommendation model by using data from checkout and the 360° customer view, and another team uses this model and focuses to present the calculated recommendations in the online shop or in promotional emails.

Consumer-aligned

In a company, there are also business departments that need data from the whole value stream to make sensible decisions, with people working in these departments are business experts but not engineers or technology-savvy. Management and controlling requires detailed reports and KPIs from all domains to identify strengths and deviations. Marketing does funnel and web analysis over all steps in the customer journey in their own optimized tools, such as Google Analytics or Adobe Analytics. In these domains, the data model is optimized for a specific department's needs and can therefore be described as **consumer-aligned**. Consumer-aligned reports were often one of the main tasks of central data teams. With data mesh, (new) consumer-aligned domain teams focus on fulfilling data needs of one specific business domain, allowing them to gain deep domain knowledge and constantly develop better analytical results. Business and IT grow closer together, either by building integrated domain teams or by having engineering teams that provide domain data as a service for the business, e.g., to support C-level or controlling. Their data are typically used for their analytics and reports, but does not need to be published and managed as data products for other domains.

Tech Stacks

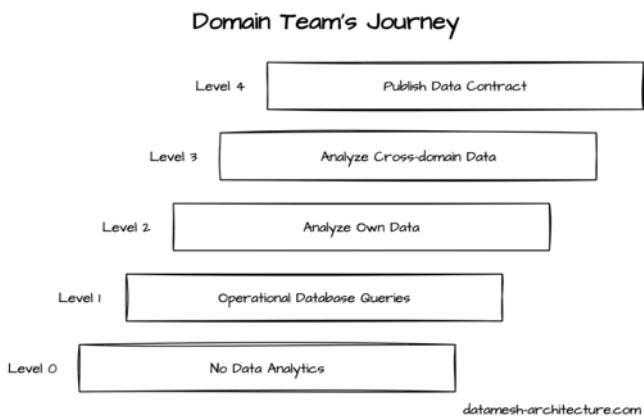
Data mesh is primarily an organizational approach, and that's why you can't buy a data mesh from a vendor. Technology, however, is important still as it acts as an enabler for data mesh, and only useful and easy to use solutions will lead to domain teams' acceptance. The available offerings of cloud providers already provide a sufficient set of good self-serve data services to let you form a data platform for your data mesh. We want to show which services can be used to get started.

There are a lot of different ways to implement a data mesh architecture. Here is a selection of typical tech stacks that we saw:

- Google Cloud BigQuery
- AWS S3 and Athena
- Azure Synapse Analytics
- dbt and Snowflake
- Databricks
- MinIO and Trino
- Starburst Enterprise (TBD)

If you want to share your tech stack here, feel free to [reach out to us](#).

Domain Team's Journey

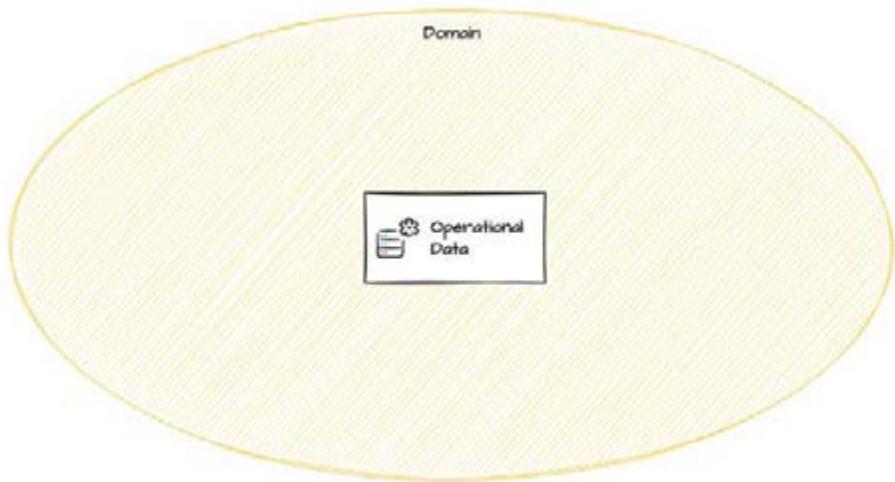


Just as the data team has a journey to go on, each of your domain teams has to go on a journey to become a contributing part of your data mesh as well. Each team can start their journey whenever they are ready and at their own pace. The benefits arise already along the journey. Teams will quickly gain from first data-driven decisions, starting an avalanche to use more and better data for even deeper insights. The data mesh evolves with each team that shares their data as products, enabling data-driven innovation.

To make this journey successful, the team needs three things: a clear data mesh vision from top management to get everybody moving in the same direction, a supportive environment including an easy-to-use self-serve data platform to get the engineering team on a learning path toward data analytics, and a high trust environment to walk the journey in their own way and pace.

So let's start your journey!

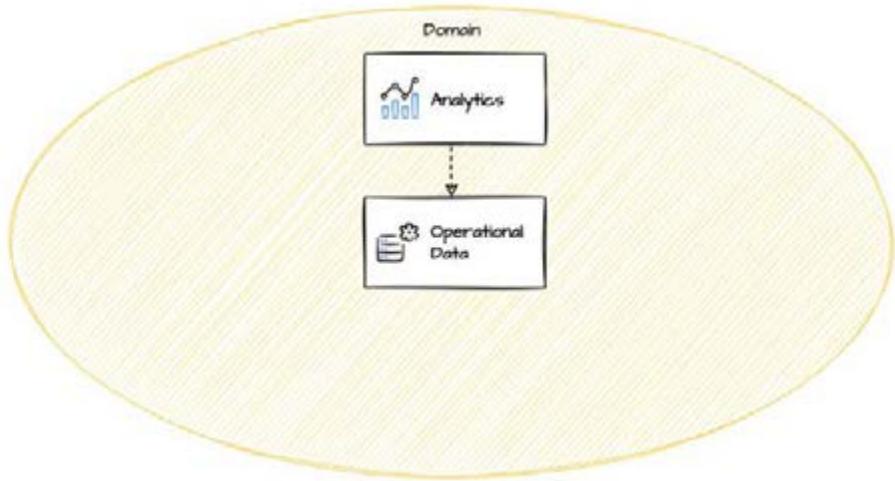
Level 0 No Data Analytics



Your team is responsible for a domain and builds and operates self-contained systems ↗ including the necessary infrastructure. It was quite an effort to build these systems, and you were highly focused on delivery excellence. These operational systems now generate domain data.

Data analytics was just not relevant.

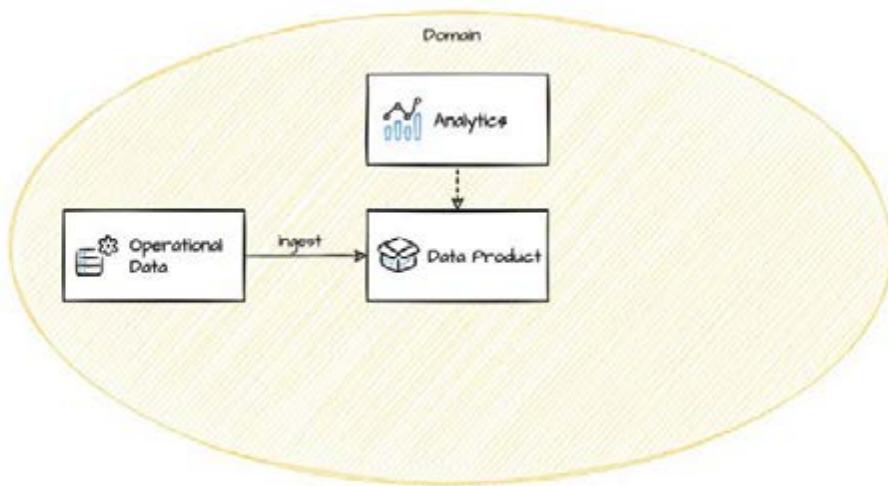
Level 1 Operational Database Queries



Being in production, you probably have to investigate an incident and need to analyze how many customers are affected. Also, some stakeholders might have questions regarding your data, such as "Which in-stock articles haven't been sold in the last six months?" or "What were the shipping times during the last Black Week?" To answer all these questions, you send analytical queries to your operational database. Over time, you also do some first explorative analytics to get a deeper understanding of your system's behavior.

This increases load on your production database, and you might be tempted to change the production database to better support your analytical queries, like creating additional indices. You might offload the additional load to read replicas. But analytical queries are still slow and cumbersome to write.

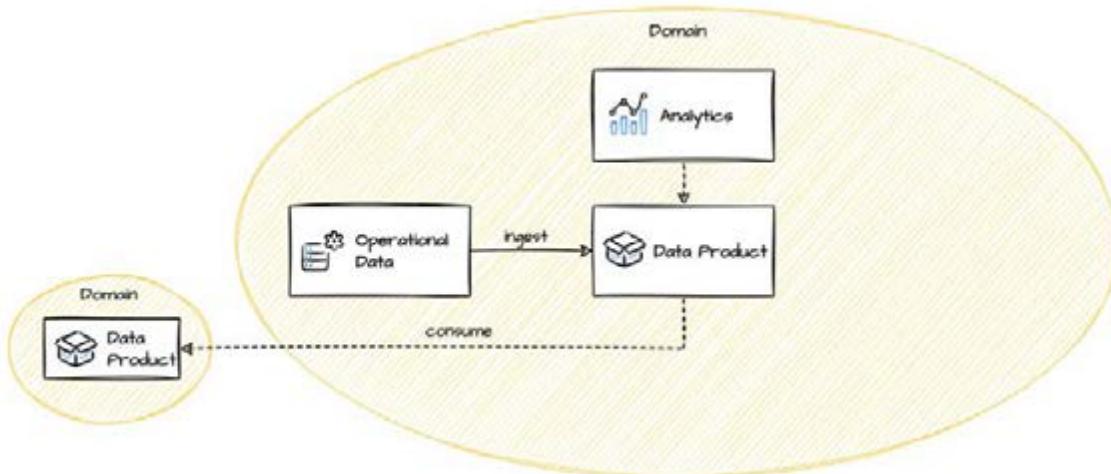
Level 2 Analyze Own Data



With the pains of slow and hard-to-write analytical queries in the back of your mind, you try out the self-serve data platform that's being promoted by the data platform team. For example, you now have access to Google BigQuery. On this platform, your team starts to build an analytical data model by ingesting messages from Kafka. This is your first data product. The data platform allows you to analyze data covering your own systems with maintainable and fast queries, while keeping the schemas of your operational databases untouched. You learn how to structure, preprocess, clean, analyze, and visualize analytical data—that's a lot to learn even though most is SQL, which you are already familiar with.

As questions regarding your own data can now be answered quickly, you and your product owner now enter the cycle of making data-driven decisions: define hypotheses and verify with data.

Level 3 Analyze Cross-domain Data

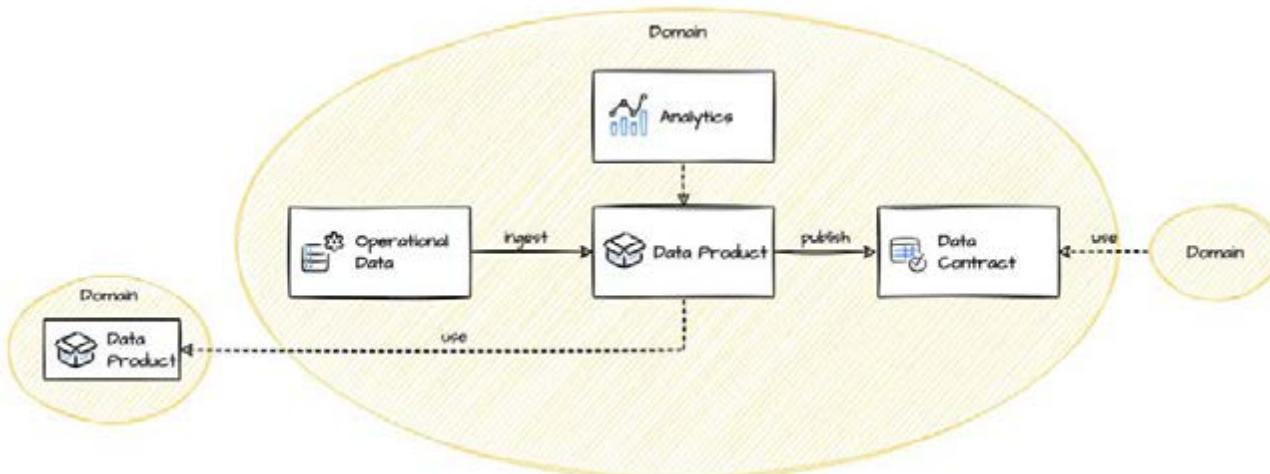


Analyzing your own domain data is a great start, but combining it with data from other domains is where the magic begins. It allows you to get a comprehensive view despite the decentralization of data. Examples are A/B tests of the effect of a UI change to the conversion rate or building up machine learning models for fraud detection that include previous purchasing history and current click stream behavior. This requires that other teams share their data products in a way that your team can discover, access, and use it. This is when the mesh begins to form itself.

When a team becomes a consuming member of the data mesh, it starts to gain interest in the interoperability and governance of the data mesh. Ideally, the team will send a representative to the data mesh governance group.

In case you are the first team, you may have to skip this step for now and move on to level 4 and be the first to provide data for others.

Level 4 Publish Data Contracts



Based on other teams' needs, you share your data with others by publishing data contracts. For example, you provide the confirmed, rejected, and aborted orders so others can correlate their events to the conversion rate. Instead of just being a consumer of data products, you become a publisher of data products. You generate value for other teams. But at the same time, it increases your responsibility and operational duties in the long term.

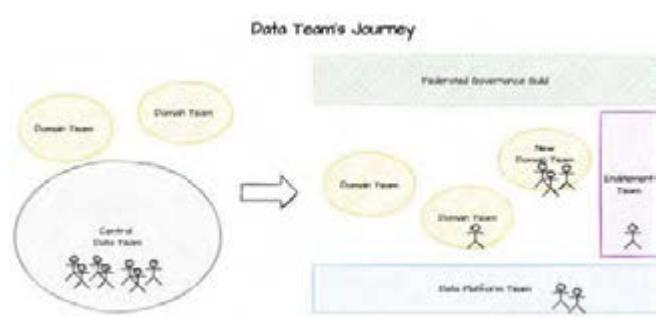
Published data products must comply with the global policies defined by the federated governance group. You have to know and understand the current global policies. Now, at the latest, you need to participate in and contribute to the federated governance group.

Data Team's Journey

Data mesh is primarily an organizational construct and fits right into the principles of team topologies [↗](#). It shifts the responsibilities for data toward domain teams which are supported by a data platform team and a data enabling team. Representatives of all teams come together in a federated governance group to define the common standards.

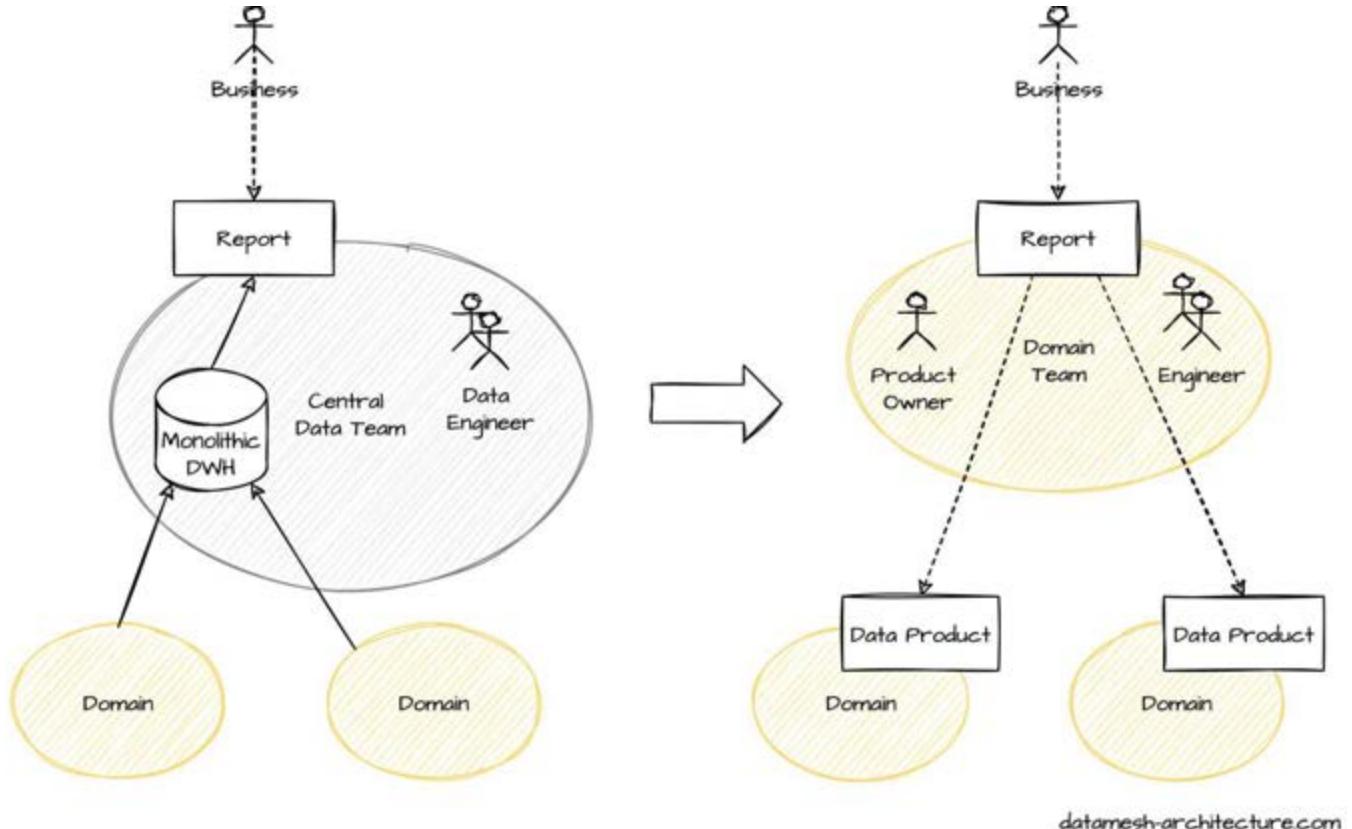
Today, in many organizations a central data team is responsible for a wide range of analytical tasks, from data engineering and managing data infrastructure to creating C-level reports. Such a central data team suffers from cognitive overload, including domain, technical, and methodical knowledge. Data mesh mitigates this.

Data mesh offers new perspectives for members of the central data team as their analytical and data engineering skills remain highly necessary. For example, they are a perfect fit to establish the data platform for



people that prefer to work on the infrastructure. Some of them can form a data enabling team to act as internal consultants, helping domain teams on their journey.

Regardless of their new roles, many of them will meet again in the data mesh federated governance group to shape the future of the data mesh.



datamesh-architecture.com

The real mind shift, however, happens when founding new data-centric domains as shown in the figure above. Let's look at typical management reports that large central data teams usually produce based on monolithic data warehouses or data lakes. With data mesh, the data engineers who created those management reports build a new domain team together with a dedicated product owner. As engineers of the new domain team, they now can focus on their new domain and their consumers. This allows them to gain deep domain knowledge over time, resulting in better reports and continuous optimizations. In addition, they switch from using that monolith data warehouse to data products from other domains. This switch is a gradual process driven by the demand for data products, accelerating the forming of a data mesh. The product owner negotiates with other domain teams about the required data products and makes sure that the reports and other products the new domain team will build in the future fulfill the needs of the business.

As existing domain teams on their journey do more and more data analytics, another perspective for members of the central data team is to join one of those teams. With their existing knowledge, they can accelerate the domain teams' journeys toward a data mesh by

spreading and teaching their knowledge and skills to the others in the team. It is important that they become full members of the team and not founding a data sub-team within the domain team. In addition to their knowledge and skills, the data engineers may also bring responsibilities and artifacts from the central data team to their domain teams. For example, customer profiling, which was previously done by the central data team, will move into the responsibility of the recommendation domain team.

The data scientists, typically, are centrally organized as well. That's why their future, organizational-wise, is quite similar to that of the central data team. The data products in the data mesh they focus on are machine learning features and models. When joining an existing domain team, such a machine learning model might be fully integrated in a microservice. So, data mesh enables such machine-learning-based services because the required MLOps ↗ capabilities can be easily built on top of the data mesh.

FAQ

So, what's really behind the hype?

Data mesh is primarily an organizational change. The responsibilities of data are shifted closer to the business value stream. This enables faster data-driven decisions and reduces barriers for data-centric innovations.

Who has actually implemented a data mesh?

There is a comprehensive collection of [user journey stories](#) from the Data Mesh Learning community that covers data mesh examples from many different industries.

Is Data Mesh for my company?

It depends, of course. There are a few prerequisites that should be in place: You should have modularized your software system following domain-driven design principles or something similar. You should have a good number (5+) of independent domain teams that have their systems already running in production. And finally, you should trust your teams to make data-driven decisions on their own.

How to get started?

Start small and agree on the big picture. Find two domain teams (that are around level 2) that have a high value use case where one team needs data from the other team. Let one team build a data product (level 4) and another team use that data product (level 3). You don't need a

sophisticated data platform yet. You can start sharing the files via AWS S3, a Git repository, or use a cloud-based database, such as Google BigQuery.

When should I avoid a Data Mesh?

There are some indicators when a data mesh approach might not be suitable for you, including:

- You are too small and don't have multiple independent engineering teams.
- You have low-latency data requirements. Data Mesh is a network of data. If you need to optimize for low-latency, invest in a more integrated data platform.
- You are happy with your monolithic highly integrated system (such as SAP). It might be more efficient to use their analytical platform.

What is Data Mesh not?

- Data Mesh is not a Silver Bullet.
- Data Mesh is not a religion.
- Data Mesh is not plug-and-play.
- Data Mesh is not a product you can just buy.
- Data Mesh is not a data-only platform.
- Data Mesh cannot be implemented by the data team alone.
- Data Mesh is not a concept for operational data.
- Data Mesh is not data virtualization.
- Data Mesh is not the successor to Data Warehouse or Data Lake.
- Data Mesh cannot be rapidly implemented as Big Bang.
- Data mesh is not a service mesh for data.
- And data mesh has absolutely nothing to do with blockchain.

Is the Data Mesh a generic solution to a distributed data architecture?

No.

By definition, data mesh does not include data products used for serving real-time needs. Data mesh focuses on analytical use cases.

Data Product: What data to include in a data product? Should a data product include other domain's data?

Data that is created and owned by a domain are prime candidates, and the domain team should be encouraged to publish them in an appropriate, cleaned and managed form.

For source-aligned domains, we mostly would argue to include reference IDs. It is OK to include other domain's data, if the data was transformed, is the basic for business decisions or the exact

state of the data at a processing time was relevant. In fact, these are cases, when the processing domain takes ownership for these data based on business cases.

Aggregate domains and consumer-aligned domains can include all foreign data that are relevant for their consumers' use cases.

What's the difference between data mesh and data fabric?

At first, [data fabric ↗](#) looks similar to data mesh because it offers a similar self-serve data platform. Looking deeper, it turns out that data fabric is a central and domain-agnostic approach, which is in strong contrast to the domain-centric and decentralized approach of data mesh. [More in this comparison article ↗](#).

What might a journey be for teams who operate commercial off-the-shelf (COTS) systems?

Many COTS systems (such as Salesforce, SAP, Shopify, Odoo) provide domain optimized analytical capabilities. So the journey for domain teams starts directly from [level 2](#).

The challenge is to integrate data products from other domains ([level 3](#), which may be skipped if not needed) and to publish data products for other domains ([level 4](#)). The system's data need to be exported to the data platform and managed as data products, conforming the global policies. As data models evolve with system updates, an anti-corruption layer is a must, e.g., as a cleaning step.

How might externally acquired datasets be part of a data mesh?

Typical examples: Price-Databases or Medical Studies. A team needs to own this dataset and bring it into the datamesh. If this is not a very technical team, the data-platform should offer an easy self-service to upload files and provide Meta-Data. An Excel API or Google Sheets might also be an option here.

How did you draw the diagrams?

We got this question quite a lot, so we are happy to share our tooling:

We use [diagrams.net ↗](#) with "Sketch" style and [Streamline Icons ↗](#). We automate the conversion to PNG, SVG and WebP with a little script.

What are your questions?

If you have any more questions, we encourage you to discuss with us on GitHub or reach out to us directly. But be warned: Your question might end up in the FAQ. :-)

Learn more



Data Mesh Learning Community

by Scott Hirleman

Scott started the Data Mesh Learning community [website](#), a [Slack channel](#), and issues a [newsletter](#) every two weeks. He collects articles and experience reports about data mesh, and even has a [podcast](#) on the topic. It is worth subscribing to stay informed.



How to Move Beyond a Monolithic Data Lake to a Distributed Data Mesh

by Zhamak Dehghani

In this article, Zhamak Dehghani introduced the idea and core principles of data mesh. It is a must-read for everyone in this field. In her [follow-up article](#), Zhamak gets into more detail.



Data Mesh

by Zhamak Dehghani

Zhamak's book about data mesh. The book not only discusses the principles of data mesh, but also presents an execution strategy.



Data Mesh

by Zhamak Dehghani

Zhamak's book about data mesh, but in German and in color. Translated by Jochen and Simon, who are co-authors of this website.



Data Mesh in Action

by Jacek Majchrzak, Sven Balnojan, and Marian Siwiak

Data Mesh in Action is a great hands-on book. We really liked the MVP hat shows how to start implementing a data mesh.



Data Mesh: Decentralized Data Analytics for Software Engineers

by Jochen Christ

Intro article to data mesh with examples in the Google Cloud.

Authors



Jochen

@jochen_christ

Jochen Christ works as tech lead at INNOQ and is a specialist for self-contained systems and data mesh. Jochen is maintainer of [HTTP Feeds](#), [Which JDK](#), and co-author of [Remote Mob Programming](#).



Larysa

@visenger

Dr. Larysa Visengeriyeva received her doctorate in Augmented Data Quality Management at the TU Berlin. At INNOQ she is working on the operationalization of Machine Learning (MLOps).



Simon

@simonharrer

Dr. Simon Harrer is a curious person working at INNOQ who likes to share his knowledge. He's a serial co-author of [Java by Comparison](#), [Remote Mob Programming](#), [GitOps](#), and, most recently, [Data Mesh](#).

Contributors

A ton of people helped us curate our content through their great feedback. Special thanks to: Anja Kammer, Benedikt Stemmhildt, Benjamin Wolf, Eberhard Wolff, Gernot Starke, Jan Bode, Jan Schwake, Julian Schikarski, Jörg Müller, Markus Harrer, Matthias Geiger, Philipp Beyerlein, Rainer Jaspert, Stefan Tilkov, Tammo van Lessen, and Theo Pack.

And if you have feedback for us as well, feel free to [discuss with us on GitHub](#) or [reach out to us directly!](#)

SUPPORTED BY



[Workshop](#) [Training](#) [Legal Notice](#) [Privacy](#)



What is a data lake?

Introduction to Data Lakes

Why do you need a data lake?
Data lake challenges
How a lakehouse solves those challenges?
authoritative data store that can power
data analytics, business intelligence and
machine learning
Building a lakehouse with
Delta Lake

Data lakes vs. data

lakehouses vs. data



Introduction to data lakes

What is a data lake?

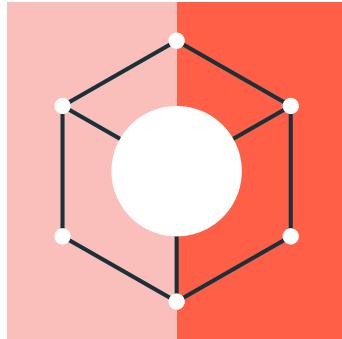
A data lake is a central location that holds a large amount of data in its native, raw format. Compared to a hierarchical data warehouse, which stores data in files or folders, a data lake uses a flat architecture and object storage to store the data. Object storage stores data with metadata tags and a unique identifier, which makes it easier to locate and retrieve data across regions, and improves performance. By leveraging inexpensive object storage and open formats, data lakes enable many applications to take advantage of the data.

Data lakes were developed in response to the limitations of data warehouses. While data warehouses provide businesses with highly performant and scalable analytics, they are expensive and proprietary and can't handle the modern use cases most companies are looking to address. Data lakes are often used to consolidate all of an organization's data in a single, central location, where it can be saved "as is," without the need to impose a schema (i.e., a formal structure for how the data is organized) up front like a data warehouse does. Data in all stages of the refinement process can be stored in a data lake: raw data can be ingested and stored right alongside an organization's structured, tabular data sources (like database tables), as well as intermediate data tables generated in the process of refining raw data. Unlike most databases and data warehouses, data lakes can process all data types — including unstructured and semi-structured data like images, video, audio and documents — which are critical for today's machine learning and advanced analytics use cases.

Why would you use a data lake?

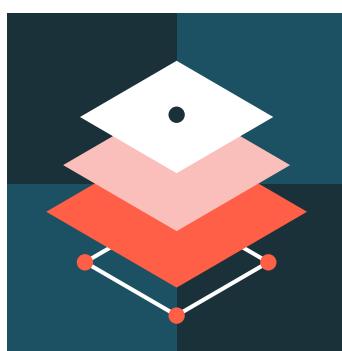
First and foremost, data lakes are open format, so users avoid lock-in to a proprietary system like a data warehouse, which has become increasingly important in modern data architectures. Data lakes are also highly durable and low cost, because of their ability to scale and leverage object storage. Additionally, advanced analytics and machine learning on unstructured data are some of the most strategic priorities for enterprises today. The unique ability to ingest raw data in a variety of formats (structured, unstructured, semi-structured), along with the other benefits mentioned, makes a data lake the clear choice for data storage.

When properly architected, data lakes enable the ability to:



Power data science and machine learning

Data lakes allow you to transform raw data into structured data that is ready for SQL analytics, data science and machine learning with low latency. Raw data can be retained indefinitely at low cost for future use in machine learning and analytics.



Centralize, consolidate and catalogue your data

A centralized data lake eliminates problems with data silos (like data duplication, multiple security policies and difficulty with

collaboration), offering downstream users a single place to look for all sources of data.



Quickly and seamlessly integrate diverse data sources and formats

Any and all data types can be collected and retained indefinitely in a data lake, including batch and streaming data, video, image, binary files and more. And since the data lake provides a landing zone for new data, it is always up to date.



Democratize your data by offering users self-service tools

Data lakes are incredibly flexible, enabling users with completely different skills, tools and languages to perform different analytics tasks all at once.

Data lake challenges

Despite their pros, many of the promises of data lakes have not been realized due to the lack of some critical features: no support for transactions, no enforcement of data quality or governance, and

poor performance optimizations. As a result, most of the data lakes in the enterprise have become data swamps.



Reliability issues

Without the proper tools in place, data lakes can suffer from data reliability issues that make it difficult for data scientists and analysts to reason about the data. These issues can stem from difficulty combining batch and streaming data, data corruption and other factors.



Slow performance

As the size of the data in a data lake increases, the performance of traditional query engines has traditionally gotten slower. Some of the bottlenecks include metadata management, improper data partitioning and others.



Lack of security features

Data lakes are hard to properly secure and govern due to the lack of visibility and ability to delete or update data. These limitations make it very difficult to meet the requirements of regulatory bodies.

For these reasons, a traditional data lake on its own is not sufficient to meet the needs of businesses looking to innovate, which is why businesses often operate in complex architectures, with data siloed away in different storage systems: data warehouses, databases and

other storage systems across the enterprise. Simplifying that architecture by unifying all your data in a data lake is the first step for companies that aspire to harness the power of machine learning and data analytics to win in the next decade.

How a lakehouse solves those challenges

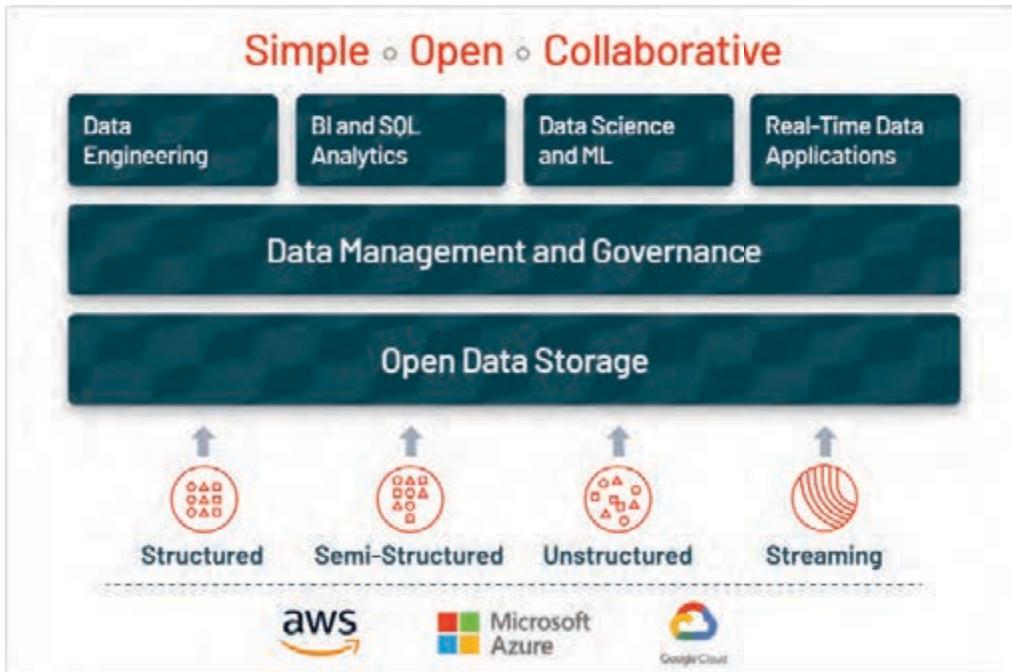
The answer to the challenges of data lakes is the lakehouse, which adds a transactional storage layer on top. A lakehouse that uses similar data structures and data management features as those in a data warehouse but instead runs them directly on cloud data lakes. Ultimately, a lakehouse allows traditional analytics, data science and machine learning to coexist in the same system, all in an open format.

A lakehouse enables a wide range of new use cases for cross-functional enterprise-scale analytics, BI and machine learning projects that can unlock massive business value. Data analysts can harvest rich insights by querying the data lake using SQL, data scientists can join and enrich data sets to generate ML models with ever greater accuracy, data engineers can build automated ETL pipelines, and business intelligence analysts can create visual dashboards and reporting tools faster and easier than before. These use cases can all be performed on the data lake simultaneously, without lifting and shifting the data, even while new data is streaming in.

Building a lakehouse with Delta Lake

To build a successful lakehouse, organizations have turned to Delta Lake, an open format data management and governance layer that combines the best of both data lakes and data warehouses. Across industries, enterprises are leveraging Delta Lake to power collaboration by providing a reliable, single source of truth. By delivering quality, reliability, security and performance on your data lake — for both streaming and batch operations — Delta Lake eliminates data silos and makes analytics accessible across the enterprise. With Delta Lake, customers can build a cost-efficient, highly scalable lakehouse that eliminates data silos and provides self-serving analytics to end users.

[Learn more about Delta Lake →](#)



Data lakes vs. data lakehouses vs. data warehouses

	Data lake	Data lakehouse	Data warehouse
Types of data	All types: Structured data, semi-structured data, unstructured (raw) data	All types: Structured data, semi-structured data, unstructured (raw) data	Structured data only
Cost	\$	\$	\$\$\$
Format	Open format	Open format	Closed, proprietary format
Scalability	Scales to hold any amount of data at low cost, regardless of type	Scales to hold any amount of data at low cost, regardless of type	Scaling up becomes exponentially more expensive due to vendor costs

	Data lake	Data lakehouse	Data warehouse
Intended users	Limited: Data scientists	Unified: Data analysts, data scientists, machine learning engineers	Limited: Data analysts
Reliability	Low quality, data swamp	High quality, reliable data	High quality, reliable data
Ease of use	Difficult: Exploring large amounts of raw data can be difficult without tools to organize and catalog the data	Simple: Provides simplicity and structure of a data warehouse with the broader use cases of a data lake	Simple: Structure of a data warehouse enables users to quickly and easily access data for reporting and analytics
Performance	Poor	High	High

Learn more about common data lake challenges →

Lakehouse best practices



Use the data lake as a landing zone for all of your data

Save all of your data into your data lake without transforming or aggregating it to preserve it for machine learning and data lineage purposes.



Mask data containing private information before it enters your data lake

Personally identifiable information (PII) must be pseudonymized in order to comply with GDPR and to ensure that it can be saved indefinitely.



Secure your data lake with role- and view-based access controls

Adding view-based ACLs (access control levels) enables more precise tuning and control over the security of your data lake than role-based controls alone.



Build reliability and performance into your data lake by using Delta Lake

The nature of big data has made it difficult to offer the same level of reliability and performance available with databases until now. Delta Lake brings these important features to data lakes.



Catalog the data in your data lake

Use data catalog and metadata management tools at the point of ingestion to enable self-service data science and analytics.

[Read the guide to data lake best practices →](#)

Shell has been undergoing a digital transformation as part of our ambition to deliver more and cleaner energy solutions. As part of this, we have been investing heavily in our data lake architecture. Our ambition has been to enable our data teams to rapidly query our massive data sets in the simplest possible way. The ability to execute rapid queries on petabyte scale data sets using standard BI tools is a game changer for us.

— Dan Jeavons, GM Data Science, Shell

[Read the full story →](#)

Resources

Delta Lake

[Delta Lake Website →](#)

[Delta Lake Demo →](#)

Webinars

[Delta Lake: The Foundation of Your Lakehouse \(Webinar\) →](#)

Documentation

Glossary: Data Lake →

Databricks Documentation: Azure Data Lake Storage Gen2 →

Databricks Documentation: Amazon S3 →

Product

Learn & Support

Solutions

Company



Worldwide

Databricks Inc.

160 Spear Street, 13th Floor

San Francisco, CA 94105

1-866-330-0121

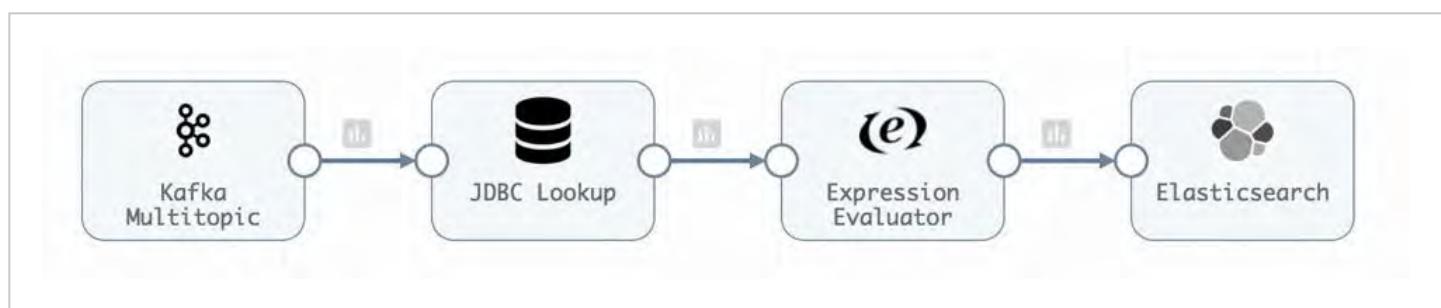
© Databricks 2023. All rights reserved. Apache, Apache Spark, Spark and the Spark logo are trademarks of the Apache Software Foundation.

Smart Data Pipelines: Tools, Techniques, and Key Concepts

How data pipelines become smart and why savvy data engineers use smart data pipelines

What Is a Data Pipeline?

A data pipeline is the series of steps that allow data from one system to move to and become useful in another system, particularly analytics, data science, or AI and machine learning systems. At a high level, a data pipeline works by pulling data from the source, applying rules for transformation and processing, then **pushing data to its destination**.

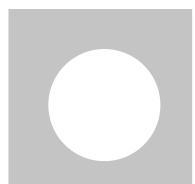


What is the Purpose of a Data Pipeline?

There's a lot of data out there. Each person creates **2.5 quintillion bytes of data per day** according to current estimates, and there are 7.8 billion people in the world. Data pipelines transform raw data into data ready for analytics, applications, machine learning and AI systems. They keep data flowing to solve problems, inform decisions, and, let's face it, make our lives more convenient.

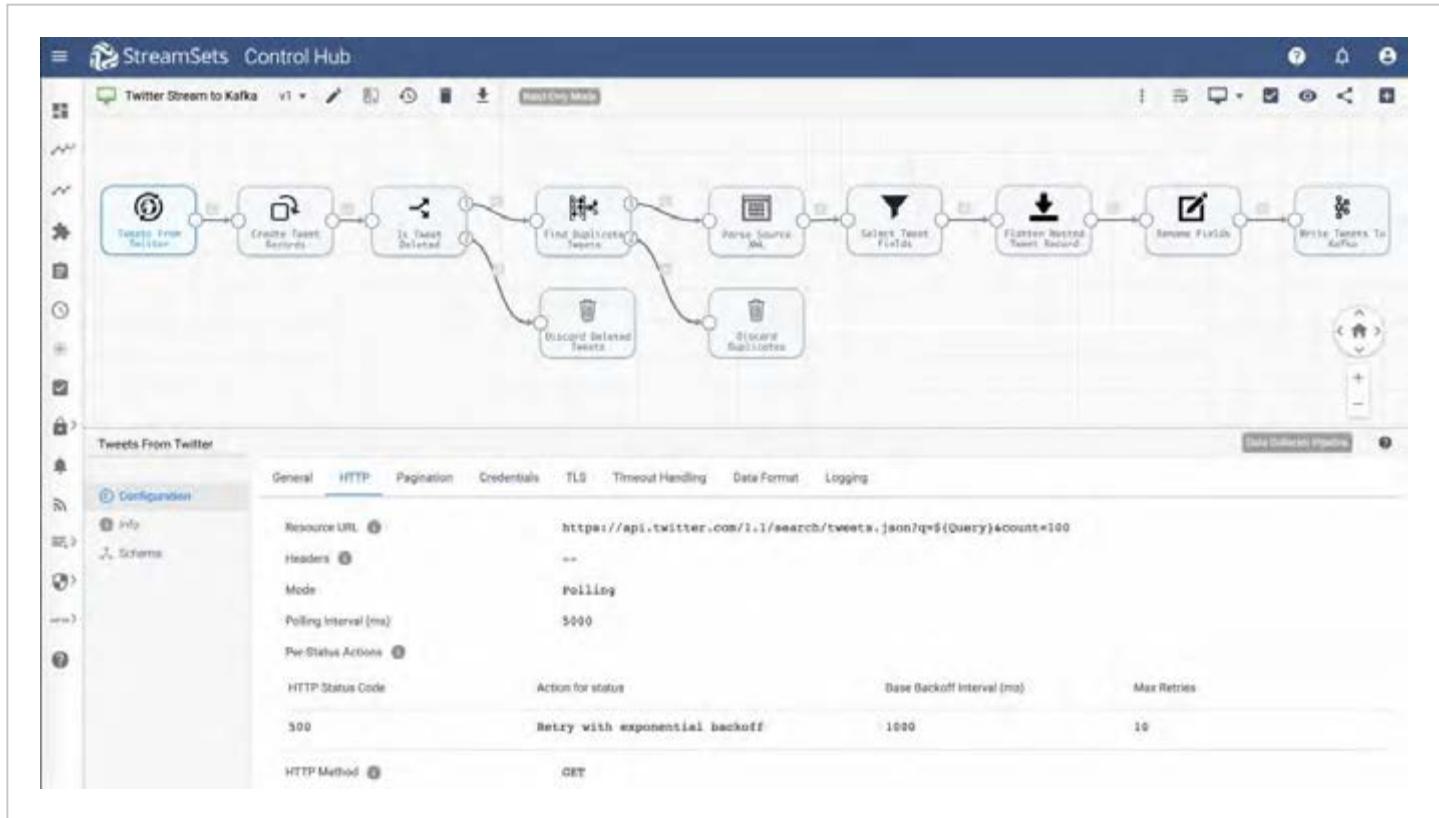
Data pipelines are used to:

- Deliver sales data to sales and marketing for **customer 360**



- Link a global network of scientists and doctors to speed drug discovery
- Recommend financial services to help a business thrive
- Track COVID-19 cases and inform community health decisions
- Combine diverse sensor data with AI for predictive maintenance

With so much work to do, data pipelines can get pretty complicated pretty fast.



Benefits of Using Data Pipelines

Data pipelines done right have incredible advantages for companies and organizations, and the work they do.

Self-service Data

Data pipelines that can be created *ad hoc* by data scientists and business analysts [unstick the IT bottleneck](#). That means when people have brilliant ideas, they can test them, fail fast, and innovate faster.

Accelerate Cloud Migration and Adoption

Data pipelines help you expand your cloud presence and [migrate data to cloud platforms](#) (yes, that's with an s). Cloud computing helps you feed many new use cases at processing speeds,

cost-effectiveness, and bursting capacity unheard of in traditional on-premises data centers. Plus your team can take advantage of rapid innovation happening on those cloud platforms such as [natural language processing](#), [sentiment analysis](#), image processing, and more.

Real-time Analytics and Applications

Real-time or near real-time functionality in consumer and business applications puts the pressure on data pipelines to deliver the right data, to the right place, right now. Streaming data pipelines deliver continuous data to [real-time analytics and applications](#).

Lifting the Lid on the Hidden Data Integration Problem

[Download Now](#)

Challenges to Using Data Pipelines

When your business depends on data, what happens when dataflow comes to a screeching halt? Or data takes a wrong turn and never reaches the intended destination? Or worse, [the data is wrong](#) with potentially catastrophic consequences?

Always Under Construction Data

Building and debugging data pipelines takes time. You have to align with the schema, set sources and destinations, check your work, find errors, and back and forth until you can finally go live, by which time the business requirements may have changed again. This is why so many data engineers have such a backlog of work.

Out of Order Data Pipelines

Even a small change to a row or a table can mean hours of rework, updating each stage in the pipeline, debugging, and then deploying the new data pipeline. Data pipelines often have to go offline to make updates or fixes. Unplanned changes can cause hidden breakages that take months of engineering time to uncover and fix. These unexpected, unplanned, and unrelenting changes are referred to as “[data drift](#)”.

Build It and They Will Come

Data pipelines are built for specific frameworks, processors, and platforms. Changing any one of those infrastructure technologies to take advantage of cost savings or other optimizations can mean weeks or months of rebuilding and testing pipelines before deployment.

Before we look at how to address these challenges to data pipeline development, we need to take a moment to understand how data pipelines work.

How Does a Data Pipeline Work?

When a data pipeline is deployed and running, it pulls data from the source, applies rules for transformation and processing, then pushes data to its destination.

5 Common Data Pipeline Sources

- JDBC
- Oracle CDC
- HTTP Client
- HDFS
- Apache Kafka

Data sources handle data in very different ways and might include applications, messaging systems, data streams, relational and NoSQL databases, cloud object storage, [data warehouses](#), and [data lakes](#). The data structure varies significantly, depending on the source.

Common Transformations

Transformations are the changes to data structure, format, or values as well as calculations and modifications to the data itself. A pipeline might have any number of transformations embedded to prepare data for use or route it correctly. A few examples:

- Masking PII (personally identifiable information) for protection and compliance
- [Converting data type for fields](#)
- Calculating based on a formula or expression

- [Renaming fields, columns, and features](#)
- [Joining or merging datasets](#)
- [Converting data formats \(JSON to Parquet, for example\)](#)
- [Generating Avro Schema](#) and other schema types on the fly
- [Handling Slowly Changing Dimensions](#)

5 Common Data Pipeline Destinations

- Apache Kafka
- JDBC
- Snowflake
- Amazon S3
- Databricks

Destinations are the systems where the data is ready to use, put directly into use, or stored for potential use. They include applications, messaging systems, data streams, relational and NoSQL databases, data warehouses, data lakes, and cloud object storage.

Most data pipeline engineering tools offer a [library of connectors and integrations](#) that are pre-built for fast pipeline development.

Data Engineer's Handbook 4 Cloud Design Patterns

[Download Now](#)

Data Pipeline Architectures

Depending on the type of data you are gathering and how it will be used, you might require different types of data pipeline architectures. Many [data engineers consider streaming data](#)

[pipelines the preferred architecture](#), but it is important to understand all 3 basic architectures you might use.

Batch Data Pipeline

Batch data pipelines move large sets of data at a particular time or in response to a behavior or when a threshold is met. A batch data pipeline is often used for [bulk ingestion or ETL processing](#). A batch data pipeline might be used to deliver data weekly or daily from a CRM system to a data warehouse for use in a [dashboard for reporting and business intelligence](#).

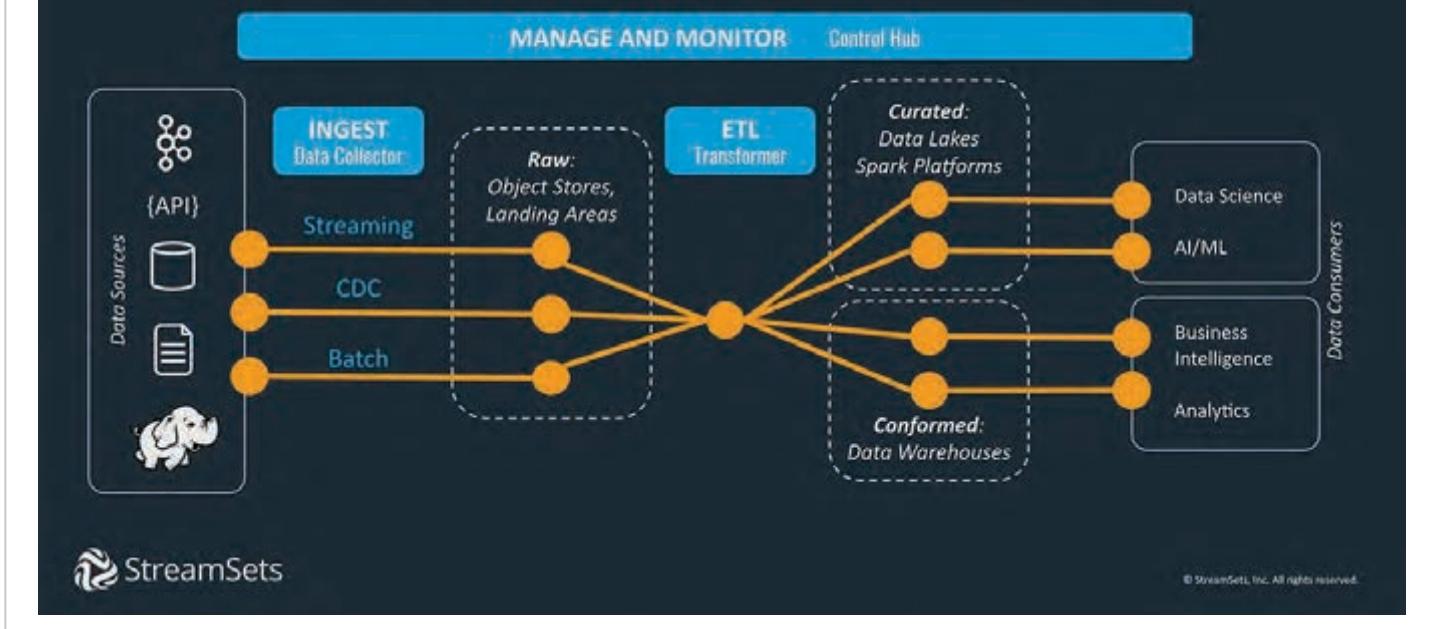
Streaming Data Pipeline

Streaming data pipelines flow data continuously from source to destination as it is created. Streaming data pipelines are used to [populate data lakes](#) or as part of [data warehouse integration](#), or to publish to a messaging system or data stream. They are also used in event processing for real-time applications. For example, streaming data pipelines might be used to provide real-time data to a [fraud detection system](#) and to monitor quality of service.

Change Data Capture Pipeline (CDC)

Change data capture pipelines are used to refresh data and keep multiple systems in sync. Instead of copying the entire database, only changes to data since the last sync are shared. This can be particularly useful [during a cloud migration project](#) when 2 systems are operating with the same data sets.

Modern Data Integration: Data Engineering



Data Pipeline Tools

Building a single pipeline for a single purpose at a given time is no problem. Grab a simple tool for setting up a data pipeline or hand code the steps. But how do you scale that process to thousands of data pipelines in support of increasing demand for data across your organization, over months or years? When considering data pipeline tools, it is important to think ahead to where your data platform is headed.

- Are you grabbing data from one place to put it somewhere else? Or do you need to transform it to fit the downstream analytics requirements?
- Is your data environment stable and fully under your control? Or is it dynamic and pulling data from systems or apps outside of your control?
- Will the pipeline move data once, for a short-term analysis project? Or will the pipelines you build need to be operationalized to handle data flows over time?

Data Ingestion and Data Loading Tools

Data ingestion and data loading tools that make data pipelines easy to set up, easy to build, and easy to deploy solve the “under construction” issue, but only if you can count on your data scientists to do the data prep work later. These tools only support the most basic transformations and work best for simply copying data. Instead of being intent-driven, these data pipelines are rigid and embedded with the specifics of data structure and semantics.

Instead of adapting to change when [data drift](#) happens, they have to be rebuilt and deployed again.

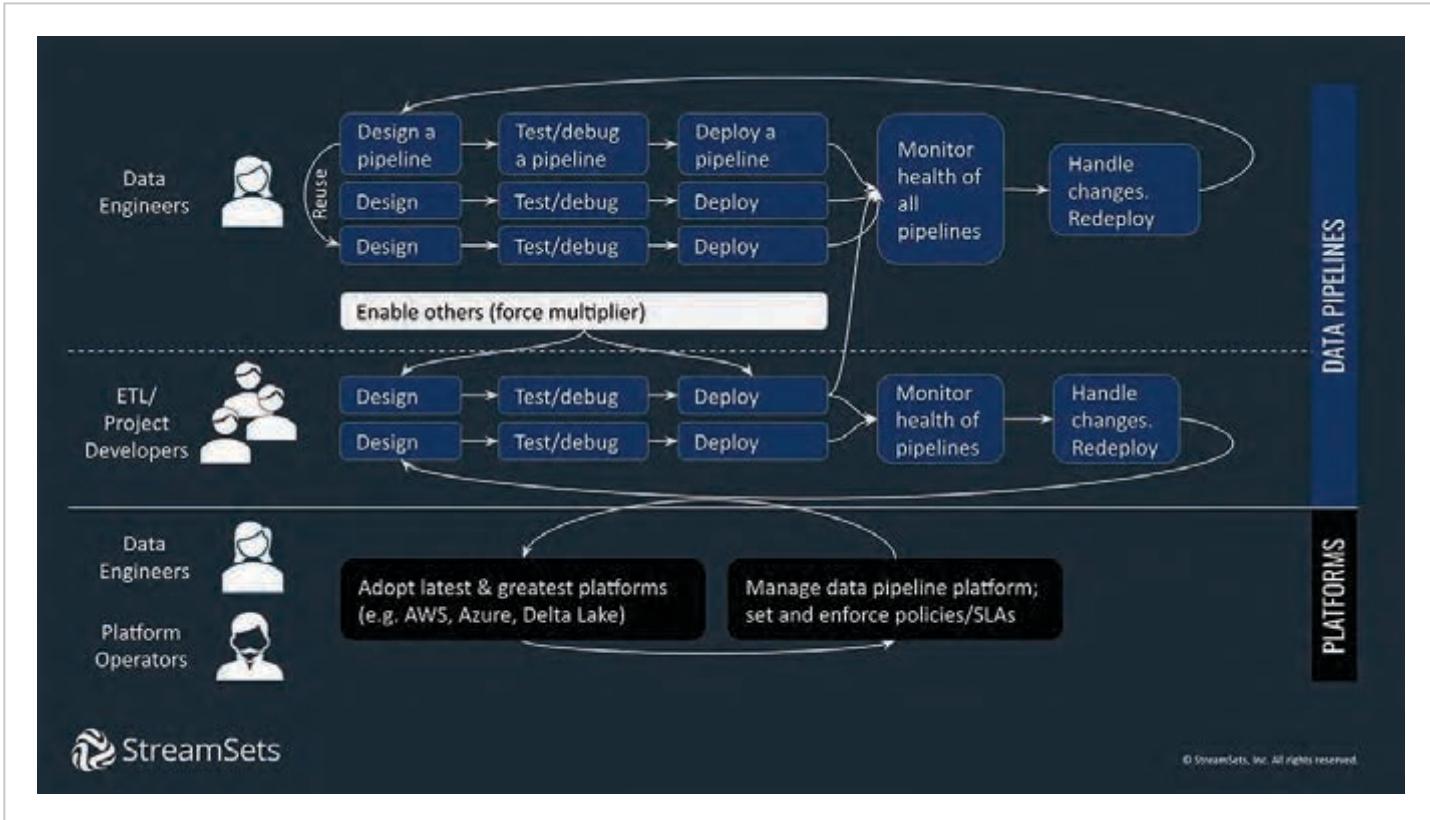
Data Integration, Data Transformation Platforms

More complex data integration or [ETL software](#) may have a solution for every possible scenario with hundreds of connectors, integrations, and transformations. But these platforms were designed for an era when data drift hardly happened. Things stayed the same for years. At the first hint of change, these data pipelines break and require massive rework. The [demands for digital transformation](#) are moving fast and planning for every possible outcome may not be possible (ahem, 2020).

Data Engineering Platforms

There is a third way. A data engineering platform [builds smart data pipelines](#) according to [DataOps](#) principles. Smart data pipelines abstract away the “how” so you can focus on the what, who, and where of the data. This is the fundamental difference between [data integration](#) and [data engineering](#). Instead of being perpetually under construction, out of order, or limited to a single platform, smart data pipelines allow you to move fast with confidence that your data will continue to flow with little to no intervention. These [dataops tools](#) allow you to:

- Design and deploy data pipelines in hours, not weeks or months
- Build in as much resiliency as possible to handle changes
- Adopt to new platforms by pointing to them, a task that takes minutes not months

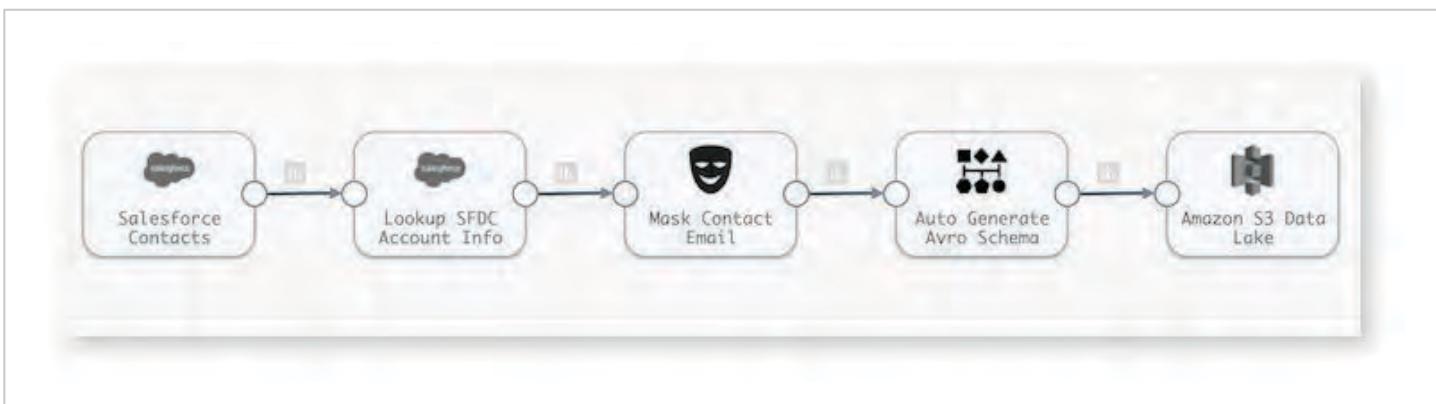


Smart Data Pipeline Examples

Batch, streaming and CDC data pipeline architectures can be applied to business and operational needs in a thousand different ways. Here are a few examples of [smart data pipelines](#) used to ingest, transform, and deliver data.

Bulk Ingestion from Salesforce to a Data Lake on Amazon

This [bulk ingestion data pipeline](#) would be ideal for archiving Salesforce contacts with some account information in Amazon S3. A Salesforce data ingestion pipeline might run in batch mode for periodic archiving or in real-time to constantly offload customer data. The destination could be any cloud storage platform.



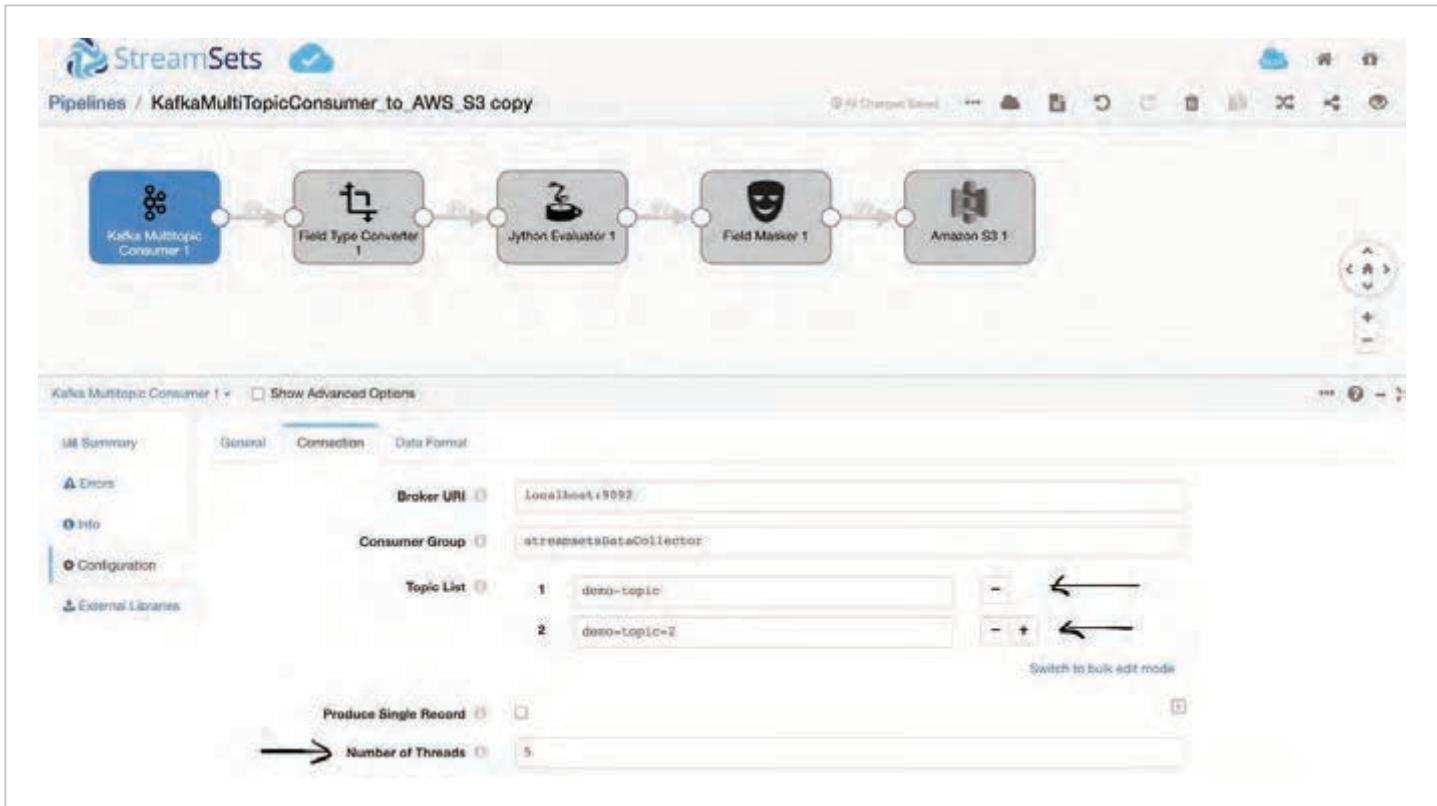
Migration to Databricks Delta Lake with Change Data Capture

Many organizations are moving data lakes from on-premises to the cloud to take advantage of pay-as-you-go pricing, higher performance, bursting capabilities, and new technologies. This [change data capture pipeline](#) tracks changes in the data source and transfers them to the destination to keep the systems in sync after the initial load.



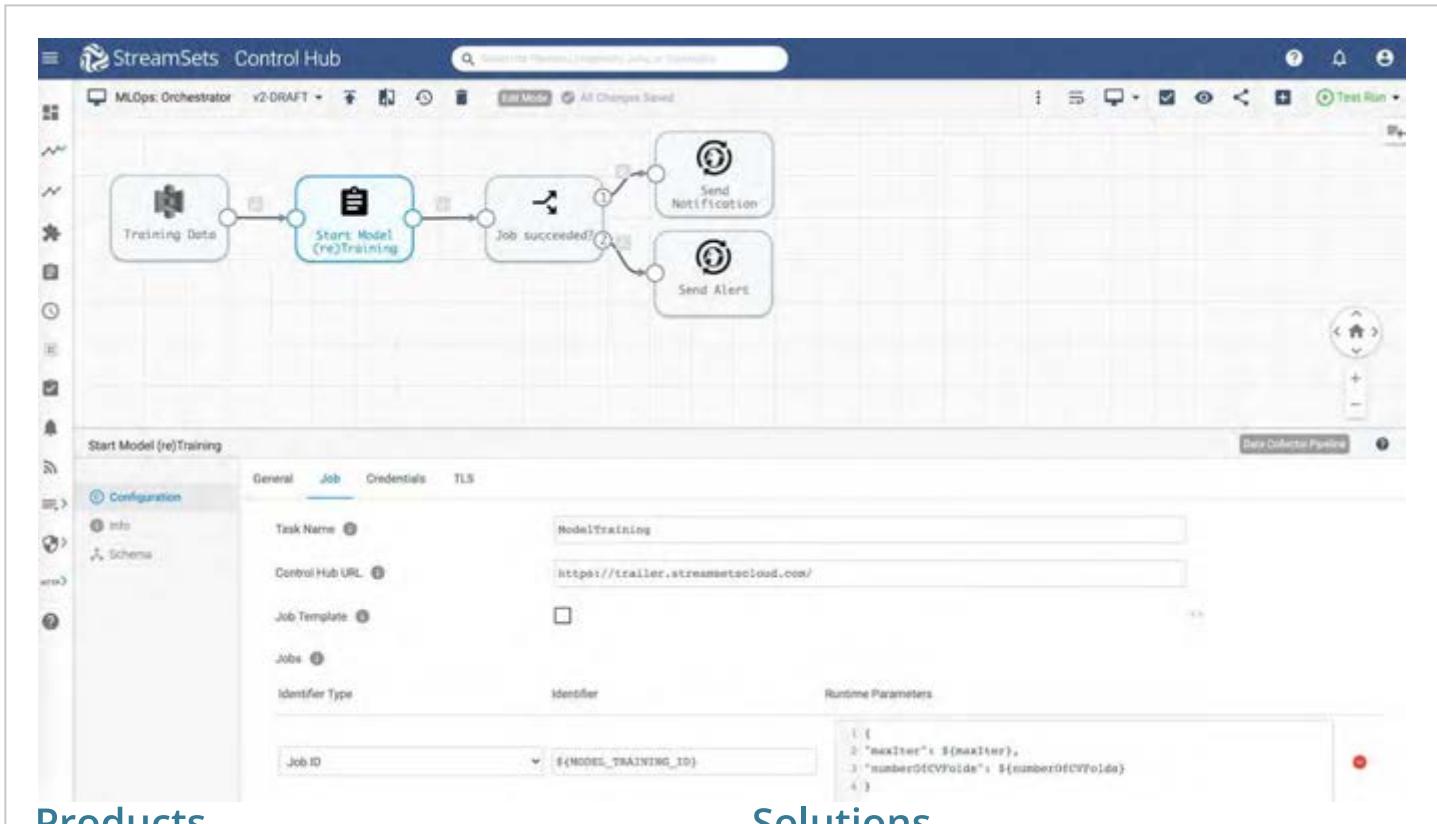
Combine Multiple Kafka Message Streams to Amazon S3

Apache Kafka is a distributed event streaming platform used for streaming analytics. This streaming data pipeline handles large amounts of data from multiple upstream applications writing to multiple Kafka topics. You can [send Kafka messages to S3](#) and scale vertically by increasing the number of threads, transforming the data and delivering it to an Amazon S3 data lake.



MLflow Integration Pipeline for Model Experiments

Machine learning models are only as good as the quality of data and the size of datasets used to train the models. Experimentation requires data scientists to create models in a rapid, iterative manner using subsets of trusted datasets. This [MLflow integration pipeline](#) on Databricks allows for tracking and versioning of model training details, plus tracking versions so data scientists have fast access to training data and can move into production more easily.



Products

StreamSets Platform
Data Collector Engine

Transformer Spark
Transformer Shuffle
Connectors

Sometimes your data has to do 2 things at once. This [Spark ETL data pipeline](#) collects sales revenue data, calculates totals by region, then delivers it to multiple destinations in the right formats to satisfy the business needs of 2 different departments in a single pipeline. Data is delivered to a Spark platform in Parquet on Azure HDInsight and to a Microsoft Power BI as SQL data. Everyone's happy.

Get Started

[Support](#)

[Academy & Certification](#)

Solutions

Agile Reporting
Cloud Data Lake Integration

Cloud Data Warehouse Integration

Real-time Analytics

Company

[Careers](#)

[Leadership](#)

[News](#)

[Software AG](#)

[Legal](#)

[Privacy Policy](#)

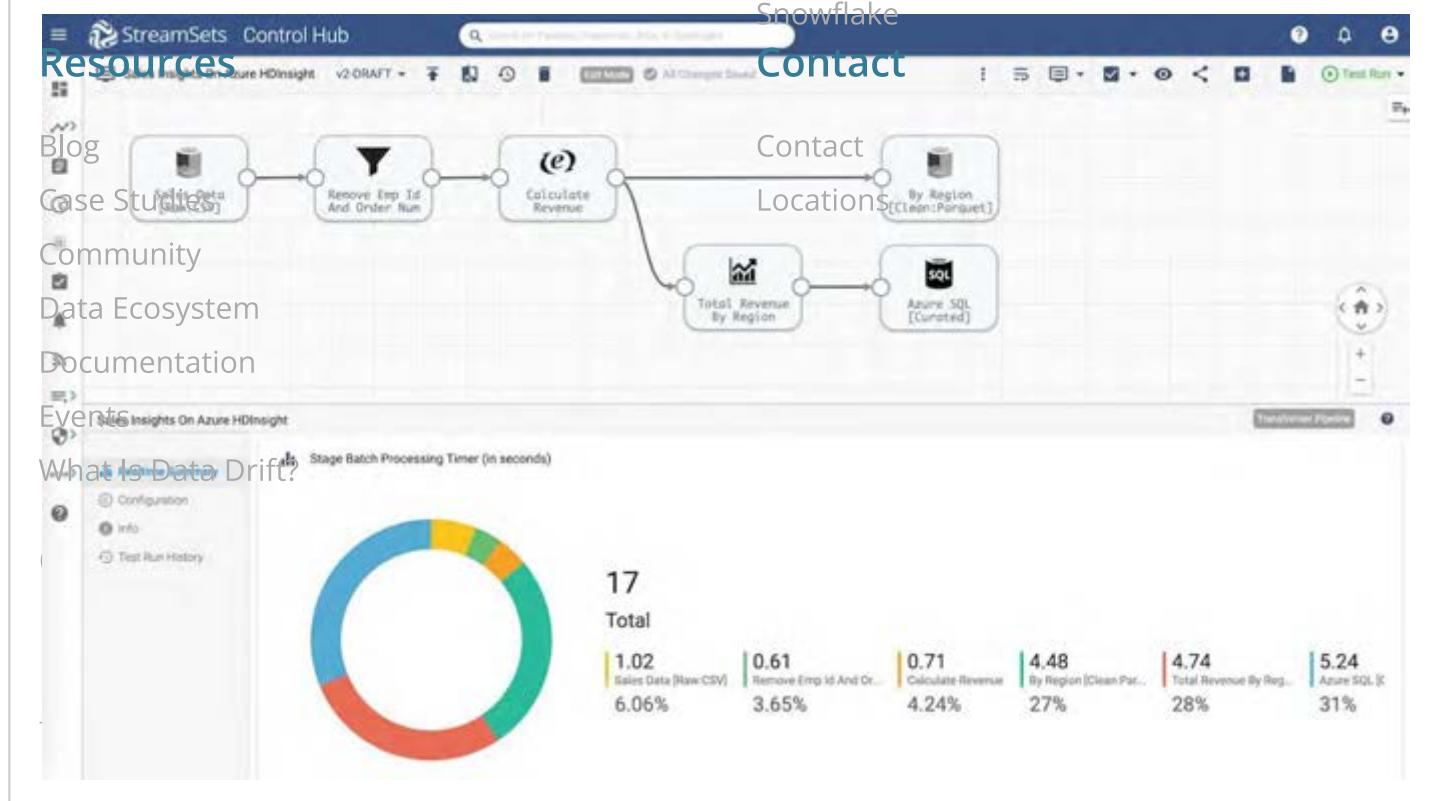
[Cookies Settings](#)

Partners

[Amazon Web Services](#)

[Databricks](#)

[Google Cloud Platform](#)



In addition to the resources above, you can always jumpstart your pipeline design with these [ready-to-deploy sample data pipelines](#). Simply, duplicate and update the sources and destinations to run.

What Is a Smart Data Pipeline?

If a [car can drive itself](#) and a [watch can notify your doctor](#) when your blood pressure goes up, why are data engineers still specifying schemas and rebuilding pipelines? A smart data pipeline is a data pipeline with intelligence built in to abstract away details and automate as much as possible, so it is easy to set up and operate continuously with very little intervention. As a result, smart data pipelines are fast to build and deploy, fault tolerant, adaptive, and self healing.

The 2020 global pandemic made it abundantly clear that companies have to be able to [respond to changing conditions quickly](#). The StreamSets [data engineering platform](#) is dedicated to building the smart data pipelines needed to [power DataOps](#) across hybrid and multi-cloud architectures. You can [build your first data pipeline](#) with StreamSets for free.

Ready to Get Started?

Build smart data pipelines for free

Try StreamSets

The Building Blocks of a Modern Data Platform

A beginner's guide to the best of breed tools and capabilities for your Data Platform initiative



Prukalpa · [Follow](#)

Published in Towards Data Science

7 min read · Feb 22, 2021

 Listen

 Share

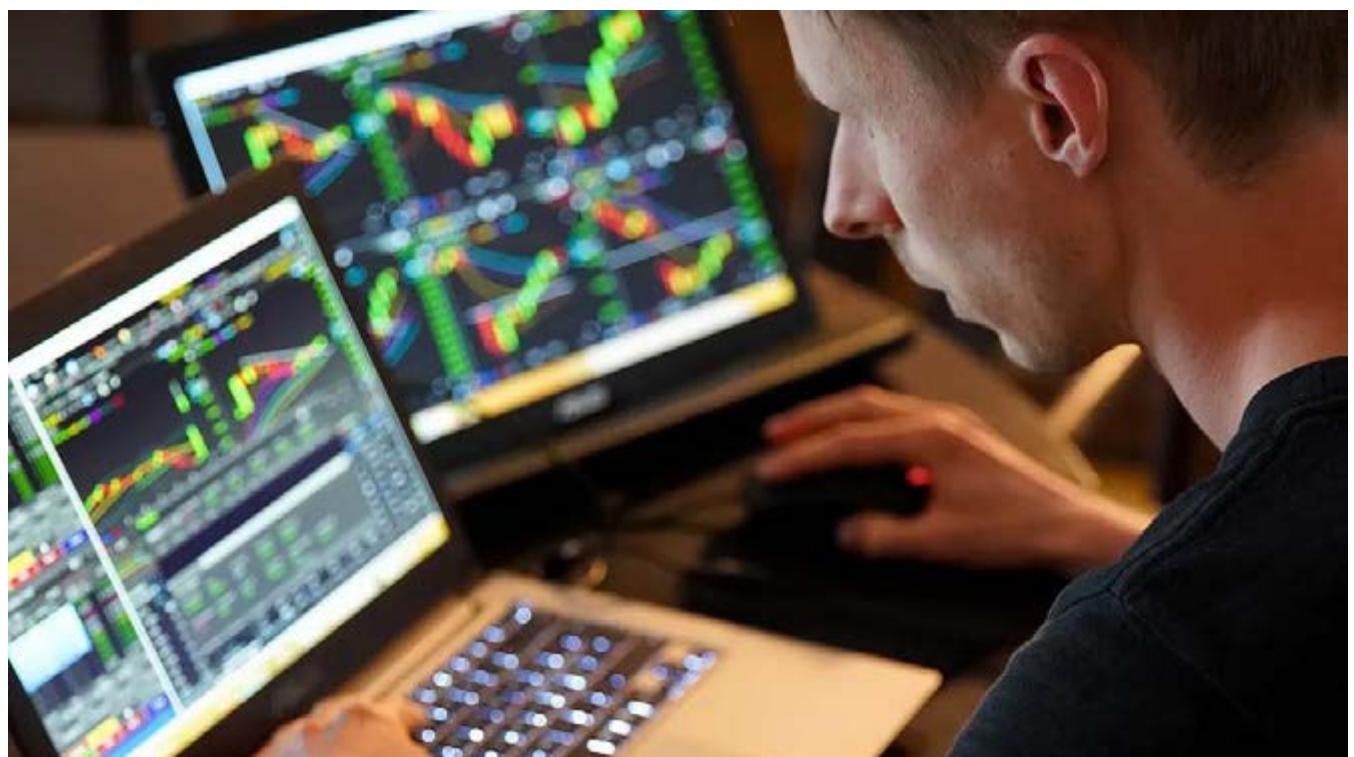


Photo by [Adam Nowakowski](#) on [Unsplash](#)

If you Google “modern data platform”, you’ll immediately be bombarded with

[Open in app](#) ↗

[Sign up](#)

[Sign In](#)



does it look like in 2021?

The short answer: a modern data platform is a collection of tools and capabilities that, when brought together, allow organizations to achieve the gold standard — a fundamentally data-driven organization.

In this article, I'll break down what a modern data platform means in practice today. This includes the three core characteristics, six fundamental building blocks, and latest data tools you should know.

The 3 characteristics of a modern data platform

Given the sheer scale and complexity of data today, it's no longer enough for a modern data platform to process and store data. It has to move and adapt faster than ever to keep up with the diversity of its data and data users. There are 3 fundamental characteristics that make a data platform truly “modern”.

Enable self-service for a diverse range of users

In a world where everyone — from business users and marketers to engineers and product managers — is an analyst, people shouldn't need an analyst to help them understand their company's data.

One key aspect of a modern data platform is that it can be used intuitively by a wide range of users. If someone wants to bring data into their work, they should be able to easily find the data they need.

This means that the platform should make it possible for all users to...

- Easily discover and analyze data within the platform
- Understand the context associated with data, such as column descriptions, history and lineage
- Derive insights from data with minimal dependencies on the data or IT team

Enable “agile” data management

One of the major challenges in legacy data platforms is their complexity. Just getting access to data usually required setting up time-consuming ETL jobs. Need to modify your data or query even a bit? The lengthy process starts all over again.

Modern data platforms aim to change that. With a well-built modern platform, data-driven decision-making should be able to move at the speed of business.

The two fundamental principles that govern modern data platforms are availability and elasticity:

- *Availability*: Data is already available in a data lake or warehouse. Modern data lakes and warehouses separate storage and compute, which makes it possible to store large amounts of data for relatively cheap.
- *Elasticity*: Compute is based on a cloud platform, which allows for elasticity and auto-scalability. For example, if end users consume the most data and analytics on Friday afternoons, it should be possible to auto-scale processing power on Friday afternoon to give users a great experience and then scale down within a few hours.

Flexible, fast set-up, and pay-as-you-go

Modern data platforms are a far cry from the complex, on-premise implementations of the Hadoop era. They are built in a cloud-first, cloud-native world, which means that they can be set up in hours, not years.

A modern platform should be...

- *Easy to set up* — no lengthy sales process, demo calls, and implementation cycles. Just login, pay via credit card, and go!
- *Pay as you go* — no up-front payments and million dollar licensing fees. The “modern” stack is all about putting power in the hands of the consumer, i.e. paying only for what you use
- *Plug and play* — the modern data stack will continue to evolve and innovate, and the best of breed tools do not enforce “lock in” like tools in the legacy era, and are instead built on open standards & APIs allowing easy integration with the rest of the stack

The key building blocks of a modern data platform

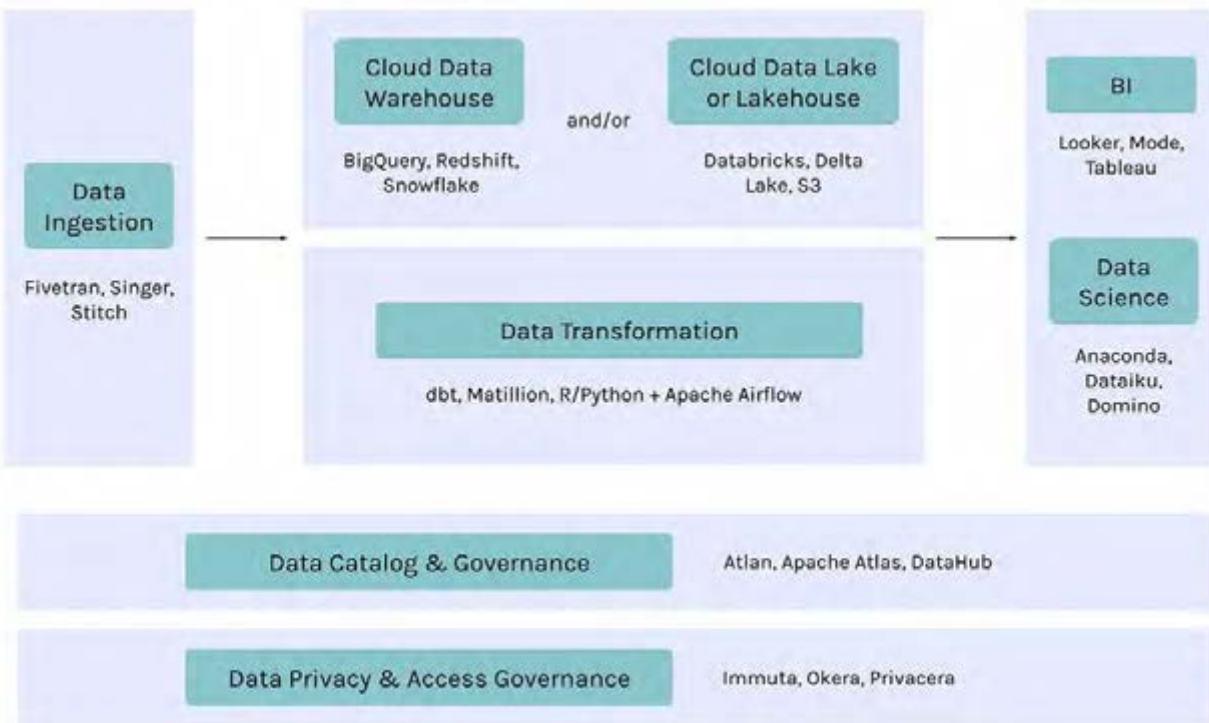


Image by [Atlan](#)

Modern data ingestion

Data ingestion is likely where your efforts to build out a modern data platform get started — i.e. how do you bring data from a variety of different data sources and ingest it into your core data storage layer?

Here are some key modern data ingestion tools:

- *SaaS tools:* [Fivetran](#), [Hevo Data](#), [Stitch](#)
- *Open-source tools:* [Singer](#), [StreamSets](#)
- Custom data ingestion pipelines built on orchestration engines like [Airflow](#)

Modern data storage and processing

The data storage and processing layer is fundamental to the modern data platform. While this architecture is evolving, we typically see 3 kinds of tools or frameworks:

Data warehouses: The cornerstone of this architecture is a modern data warehouse. These are generally the system of choice for analysts since they optimize compute and processing speed.

Some key data warehouse tools include [BigQuery](#), [Redshift](#), and [Snowflake](#).

Data lakes: A data lake architecture refers to data being stored on an object storage like Amazon S3, coupled with tools to process the data such as Spark. These cheap

storage systems are often used to store vast amounts of raw or even unstructured data.

Here are some key tools for data lakes:

- *Data storage*: [Amazon S3](#), [Azure Data Lake Storage Gen2](#), [Google Cloud Storage](#)
- *Data processing engines*: [Databricks](#) or [Spark](#); [Athena](#), [Presto](#), or [Starburst](#); [Dremio](#)

New trend alert: Data lakehouses! One of the trends we're seeing this year is the long-awaited convergence of data warehouses and lakes. This will help unify the siloed systems that most companies have created over the last decade.

For example, one concept that's emerging is the “[data lakehouse](#)” — a system design that combines the data management features such as ACID transactions and change data capture from a data warehouse with the low-cost storage of a data lake.

The Top 5 Data Trends for CDOs to Watch Out for in 2021

Modern metadata solutions, data quality frameworks, infrastructure, job roles, and other big changes are on their way.

[towardsdatascience.com](#)

Modern data transformation

There are two core implementations we're seeing today for the data transformation layer. For companies with data warehouse-first architectures, tools like dbt that leverage native SQL for transformation have emerged as the top choice for data transformation. The other common implementation is using Airflow as an orchestration engine coupled with custom transformation in a programming language like Python.

Here are some key tools for data transformation:

- *With data warehouses*: [dbt](#), [Matillion](#)
- *With an orchestration engine*: [Apache Airflow + Python](#), [R](#), or [SQL](#)

Modern business intelligence and analytics

BI dashboards have been around for ages, but the latest breed of BI and analytics tools are built to fit within a larger modern data platform. These are generally more self-service platforms that allow users to explore data, rather than just consuming graphs and charts.

Modern BI tools include [Looker](#), [Mode](#), [Redash](#), [Sigma](#), [Sisense](#), [Superset](#), and [Tableau](#).

Modern data catalogs and governance

While the modern data platform is great in some areas (super fast, easy to scale up, little overhead), it struggles with bringing discovery, trust and context to data.

As metadata itself becomes big data, we're in the midst of a leap forward in metadata management and the space is about to see significant amounts of innovation in the next 18–24 months. [I recently wrote about the idea of Data Catalog 3.0](#): a new era of software, which will be built on the premise of **embedded collaboration** that is key in today's modern workplace, borrowing principles from Github, Figma, Slack, Notion, Superhuman, and other modern tools that are commonplace today.

Here are some key tools for modern data cataloguing and governance:

- *SaaS tools:* [Atlan](#)
- *Open-source tools:* Apache [Atlas](#), LinkedIn's [DataHub](#), Lyft's [Amundsen](#)
- *In-house tools:* Airbnb's [Dataportal](#), Facebook's [Nemo](#), Uber's [Databook](#)

Data Catalog 3.0: Modern Metadata for the Modern Data Stack

It's time for a modern metadata solution, one that is just as fast, flexible, and scalable as the rest of the modern...

towardsdatascience.com

Modern data privacy and access governance

Finally, as tooling in the modern data platforms grow, a major challenge is to be able to manage privacy controls and access governance across your entire stack. While it's still early, there have been some recent players and developments in the

space and players emerging with tools that can act as an entitlement engine to apply privacy and security policies across the data stack.

Here are some key tools for data privacy and access governance:

- *SaaS services*: [Immuta](#), [Okera](#), [Privacera](#)
- *Open-source engines*: [Apache Ranger](#)

Other modern data tools you should know about

The capabilities and tools listed above are the basic layers of a modern data platform. However, each company uses data differently, so many have added additional layers and tools for specific use cases.

Real-time data processing tools

Companies that need real-time data processing generally add two additional types of tools to their data platform:

- *Real-time streaming pipelines*: [Confluent](#), [Kafka](#)
- *Real-time analytics*: [Druid](#), [Imply](#)

Data science tools

Companies that have moved past BI and analytics onto strong predictive and data science analytics often add a specific data science tool (or tools) to their data stack:

- *Preferred by data scientists*: [Jupyter Notebooks](#)
- *Other options*: [Dataiku](#), [DataRobot](#), [Domino](#), [SageMaker](#)

Event collectors

Companies with a significant digital presence often add event collectors to log and store external events. Tools like [Segment](#) and [Snowplow](#) are key additions to their data ingestion stack.

Data quality tools

This space is still relatively nascent, but it's seeing a lot of activity nowadays. Broadly, we're seeing a few aspects of data quality incorporated throughout the data stack — data quality checks during profiling, business-driven quality rules, and unit testing frameworks within the pipeline.

- *Data profiling*: This is getting engulfed by data catalog and governance tools like [Atlan](#) or open source profiling frameworks like [Amazon Deequ](#).
- *Unit testing*: Frameworks like the open-source [Great Expectations](#) are emerging and evolving, allowing unit tests to be written as a part of data pipelines itself.

Found this content helpful? I write weekly on active metadata, DataOps, data culture, and our learnings building [Atlan](#) at my newsletter, [Metadata Weekly](#). [Subscribe here.](#)

Data Platforms

Data Engineering

Data

Data Management

Data Science



Follow

Written by Prukalpa

4.8K Followers · Writer for Towards Data Science

Co-founder of Atlan ([atlan.com](#)), the active metadata platform for modern data teams | Weekly newsletter for data leaders: [metadataweekly.substack.com](#)

More from Prukalpa and Towards Data Science



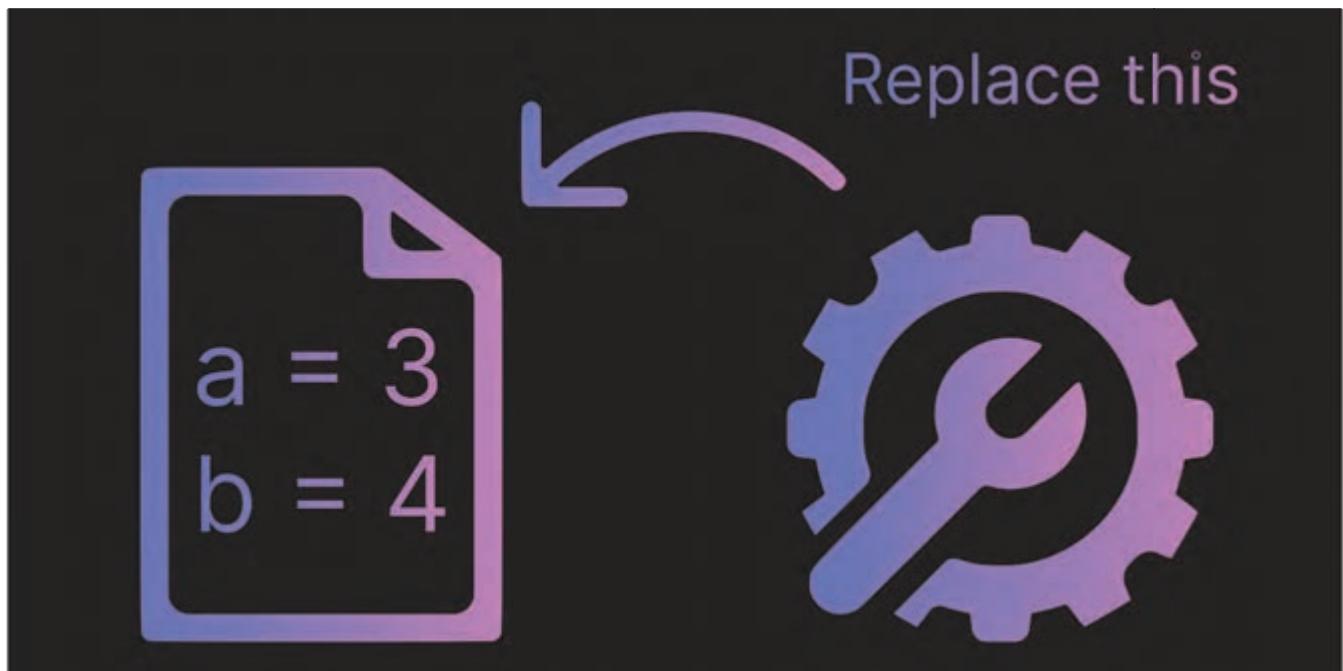
 Prukalpa in Towards Data Science

The Future of the Modern Data Stack in 2023

Featuring 4 new emerging trends and 6 big trends from last year

19 min read · Jan 13

 550  6



 Khuyen Tran in Towards Data Science

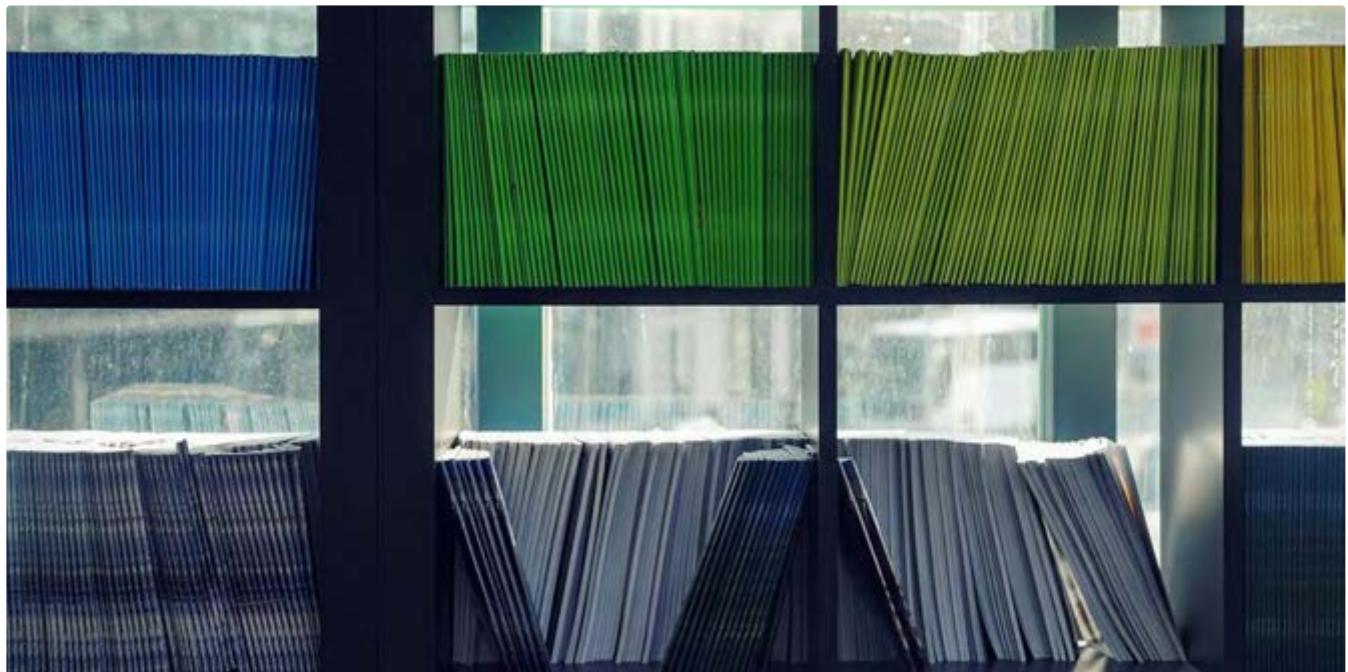
Stop Hard Coding in a Data Science Project—Use Config Files Instead

And How to Efficiently Interact with Config Files in Python

◆ · 6 min read · May 26

👏 2K

💬 21



Jacob Marks, Ph.D. in Towards Data Science

How I Turned My Company's Docs into a Searchable Database with OpenAI

And how you can do the same with your docs

15 min read · Apr 25

👏 4.3K

💬 49





Prukalpa in Towards Data Science

Data Catalog 3.0: Modern Metadata for the Modern Data Stack

It's time for a modern metadata solution, one that is just as fast, flexible, and scalable as the rest of the modern data stack.

8 min read · Jan 18, 2021



1.1K



6



See all from Prukalpa

See all from Towards Data Science

Recommended from Medium



Sven Balnojan in Geek Culture

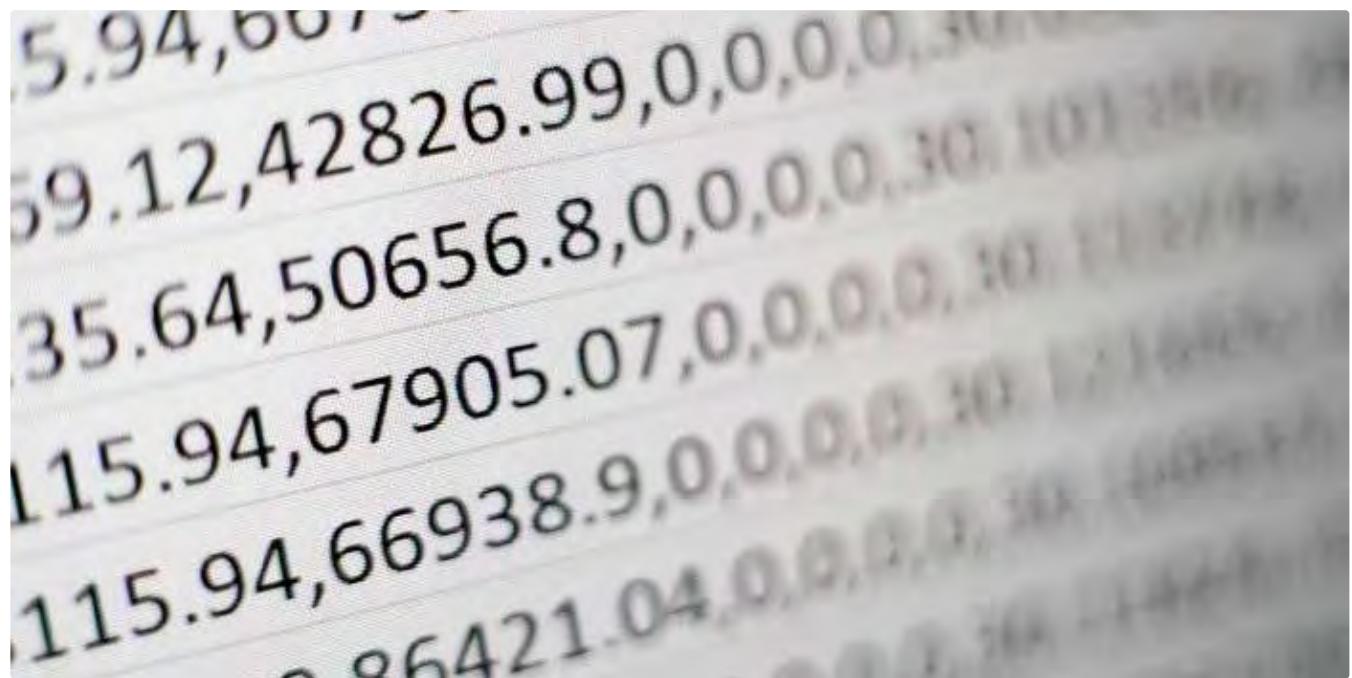
If You Only Read A Few Data Articles In 2023, Read These

The best of the best data articles on the modern data stack, data engineering best practices, building data lakes, and much more.

★ · 6 min read · Feb 6



Q 1



Saeed Mohajeryami, PhD

Implementing data Governance: A Step by Step guide for achieving compliance and data-driven...

Mastering the Fundamentals and Best Practices

★ · 12 min read · Jan 16

70



Lists



New_Reading_List

173 stories · 8 saves



Predictive Modeling w/ Python

18 stories · 73 saves



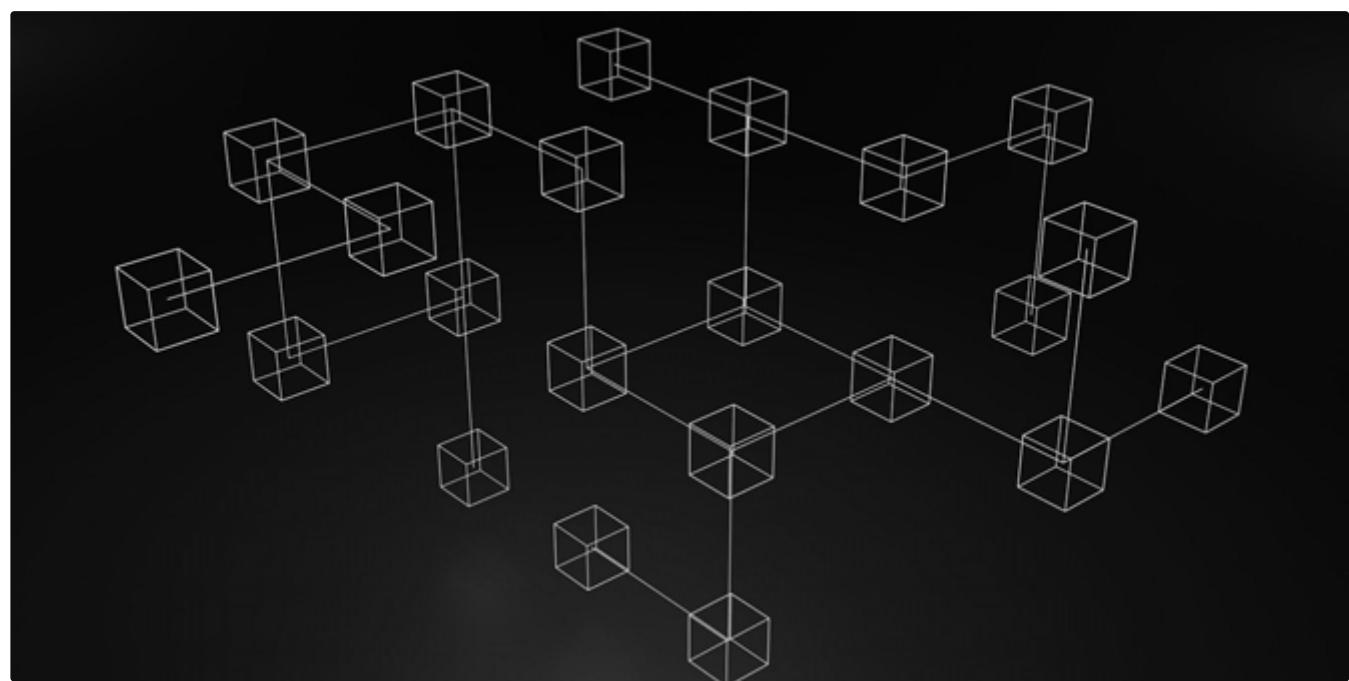
Practical Guides to Machine Learning

10 stories · 81 saves



Coding & Development

11 stories · 28 saves



Saeed Mohajeryami, PhD

Data Mesh: Concepts and best practices for implementing a Product-centric Data Architecture

Navigating the Evolution of Data Architecture

★ · 14 min read · Jan 13

👏 137

💬 2



👤 Saeed Mohajeryami, PhD in Bootcamp

Data Solution Architects: The Future of Data Management

Understanding the Role of a Data Solution Architect: From Technical to Business Responsibilities

★ · 13 min read · Jan 24

👏 43

💬 1





Eric Broda in Towards Data Science

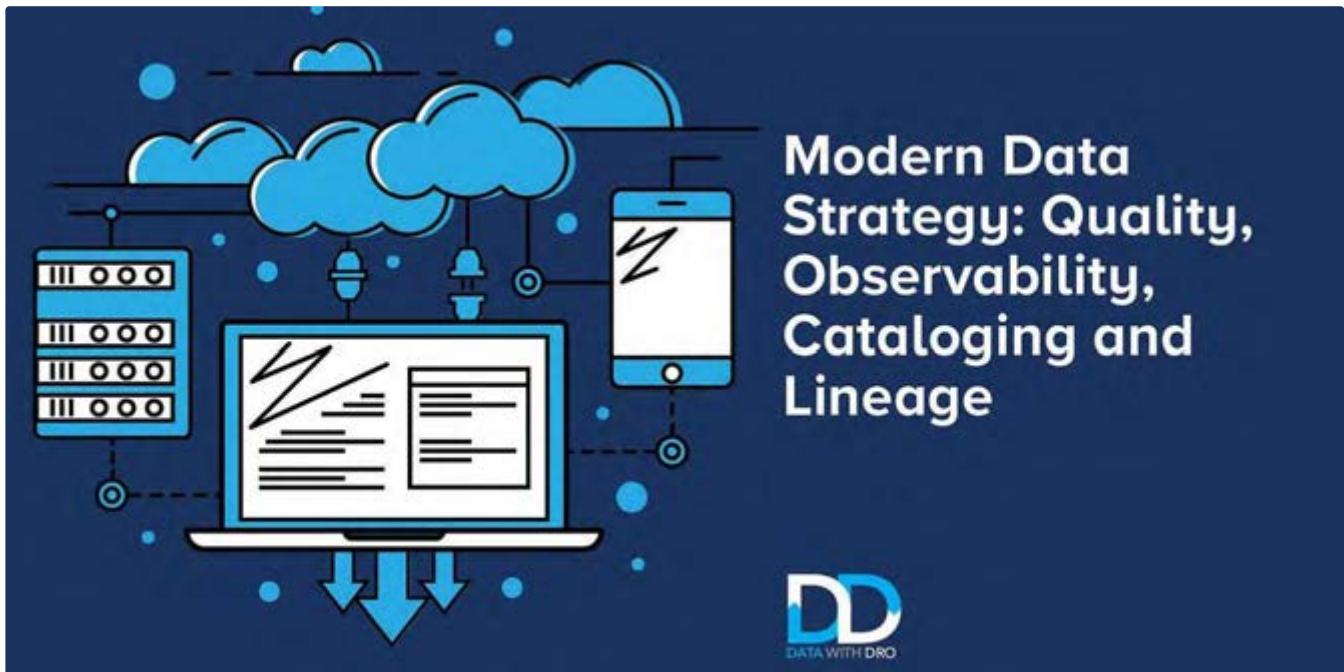
Managing a Federated Data Product Ecosystem

As Data Mesh matures, enterprises are struggling to manage their growing data product ecosystem. How can we manage this growth?

◆ · 9 min read · Jan 11

151

1



Danilo Drobac

Modern Data Strategy: Quality, Observability, Cataloging and Lineage

Simplify data management, governance and discovery to unlock the full potential of your data

★ · 7 min read · Feb 6

👏 145

💬 2



See more recommendations



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Big Data Analytics

Pravin Y Pawar

Content adapted from Big Data Analytics by Seema Acharya

Big Data Analytics

Defined

Working with datasets with huge volume and variety beyond storage and processing capability of RDBMS

Better , Faster decision in real time

Richer, faster insights into customers, partners and business

Uses Principle Of Locality to move code near to Data

BIG Data Analytics

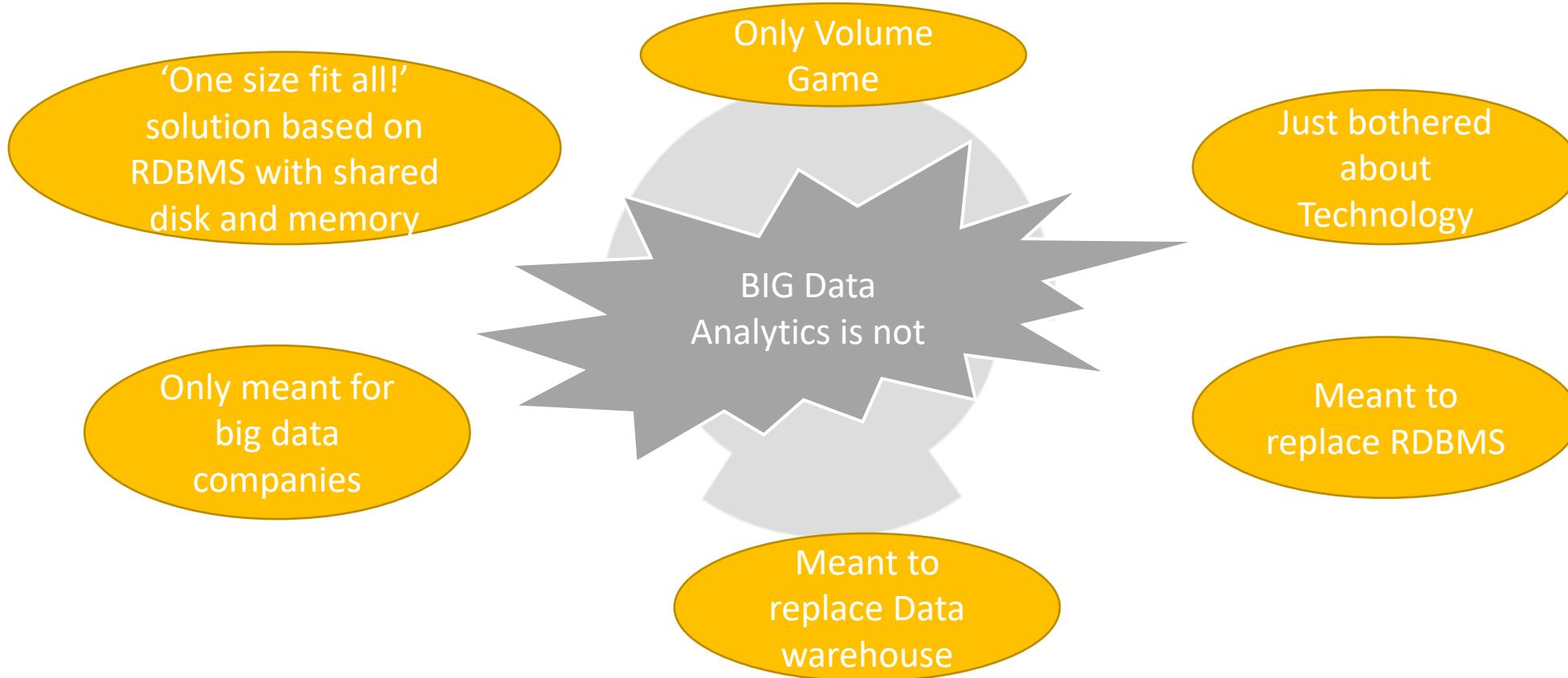
Competitive Advantage

IT's collaboration with business users and Data Scientists

Technology enabled Analytics

Support for both online and offline processing of data

!Big Data Analytics

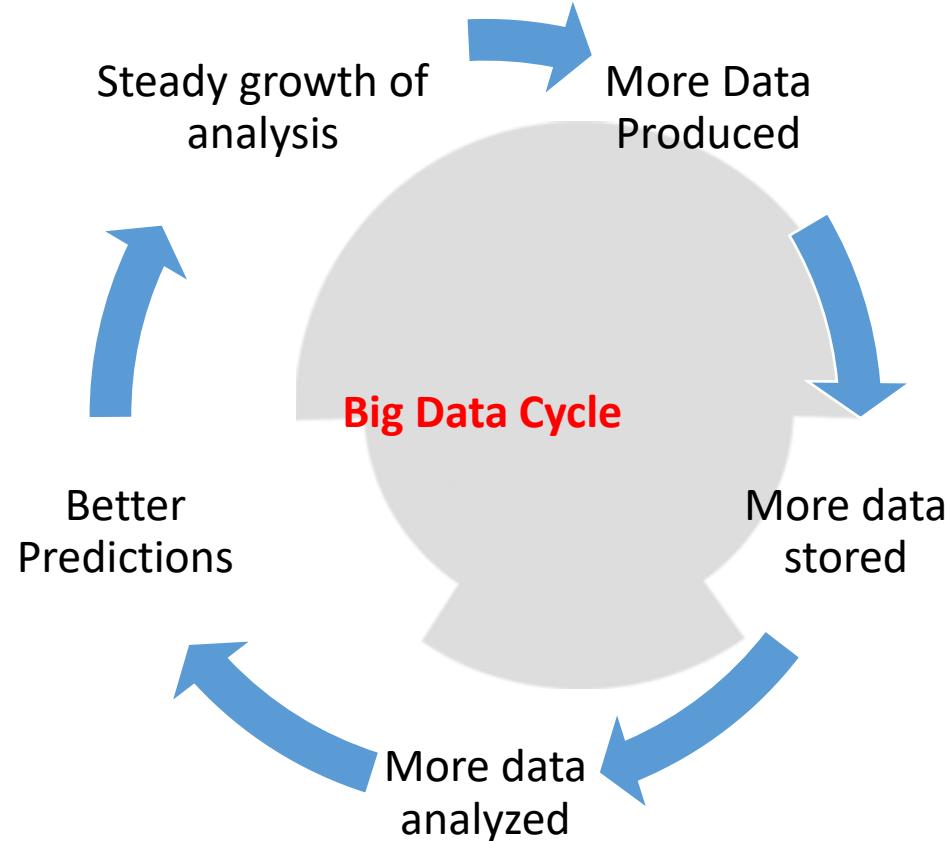


Hype?

Why there is sudden hype?

- Data is growing at 40% compound annual rate
 - ✓ Reaching 45 ZB by 2020
 - ✓ In 2010, 1.2 trillion Gigabytes data generated
 - ✓ In 2012, reached to 2.4 trillion Gigabytes
 - ✓ **Volume of world wide data expected to double every 1.2 years**
 - ✓ Every day 2.5 quintillion bytes of data is created
 - ✓ 90% of todays data is generated in last few years only!
 - ✓ Walmart processes one million customer transaction per hour
 - ✓ 500 million “tweets” are posted by users every day
 - ✓ 2.7 billion “likes” and comments by Facebooks users per day
- Cost of storage has hugely dropped
- Large number of user friendly analytics tools available for data processing

What big data entails?



Challenges preventing usage of Big Data

For the organizations

- Obtaining executive sponsorship for investments in big data and its related activities
- Getting business units to share data / information across organizational silos
- Finding right skills (Business Analysts/Data Scientists and Data Engineers) that can manage large amount of variety of data and create insights from it
- Determining approach to scale rapidly and elastically , address storage and processing of large volume, velocity and variety of Big data
- Deciding whether to use structured or unstructured, internal or external data to make business decisions
- Choosing optimal way to report findings and analysis of big data
- Determining what to do with the insights created from big data

Top challenges facing with Big Data

- Scale
 - ✓ Need is to have storage that can best withstand the onslaught of large volume, velocity and variety of data
 - ✓ Should scale vertically or horizontally?
 - ✓ RDBMS or NoSQL?
- Security
 - ✓ Most of recent NoSQL big data platforms have poor security mechanisms
 - ✓ Lack of authentication and authorization techniques while safeguarding big data
 - ✓ Big Data contains private data like credit card details, personal information, and other sensitive data , so security can not be taken lightly!

Top challenges facing with Big Data (2)

- Schema
 - ✓ Right schemas have no place
 - ✓ Technology should fit our big data ,not other way around!
 - ✓ Need is to have dynamic schema, static / fixed schemas are gone
- Continuous availability
 - ✓ Needs 24 * 7 * 365 support as data is continuously getting generated and needs to be processed
 - ✓ Almost all RDBMS , NoSQL big data platforms has some sort of downtime

Top challenges facing with Big Data (3)

- Consistency
 - ✓ Should one go for consistency or eventual consistency?
- Partition Tolerant
 - ✓ How to build partition tolerant systems that can take care of both hardware and software failures?
- Data quality
 - ✓ How to maintain data quality – data accuracy, completeness, timeliness etc.?
 - ✓ Do we have appropriate metadata in place?



Thank You!

In our next session: Technologies for Big Data Analytics



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Technologies for Big Data Analytics

Pravin Y Pawar

Technologies for Big Data Analytics

Problem of managing data

- How to manage voluminous, varied and scattered data?
 - ✓ Traditional data storage and processing systems not able to cope up with this type of data
 - ✓ Prompted the requirement of systems for storage and processing of big data
- Most effective and popular ones are
 - ✓ Distributed and parallel processing
 - ✓ Hadoop
 - ✓ Big Data Cloud
 - ✓ In-memory computing

Popular Technologies

Three options

- Big Data Clouds
 - ✓ Helps to save cost and better management of resources
 - ✓ Everything based on a services model
- Hadoop
 - ✓ Most popular open source platform for big data processing
 - ✓ Used by data-driven organizations to derive value which can be used in decision making
- In-memory Computing (IMC)
 - ✓ By usage of main memory (RAM) helps to manage data processing tasks better and faster
- **Choice depends heavily on the requirements!**



Thank You!

In our next session: Distributed and Parallel Computing for Big Data



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

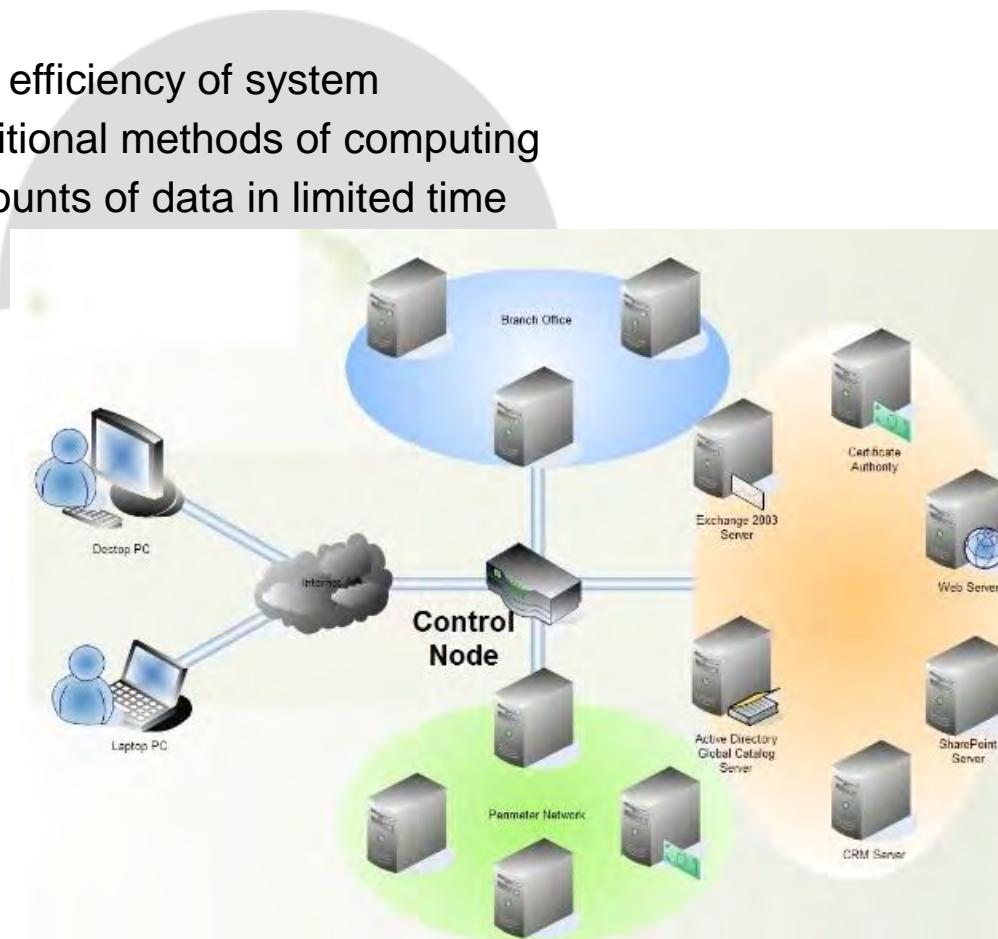
Distributed and Parallel Computing for Big Data

Pravin Y Pawar

Distributed Computing

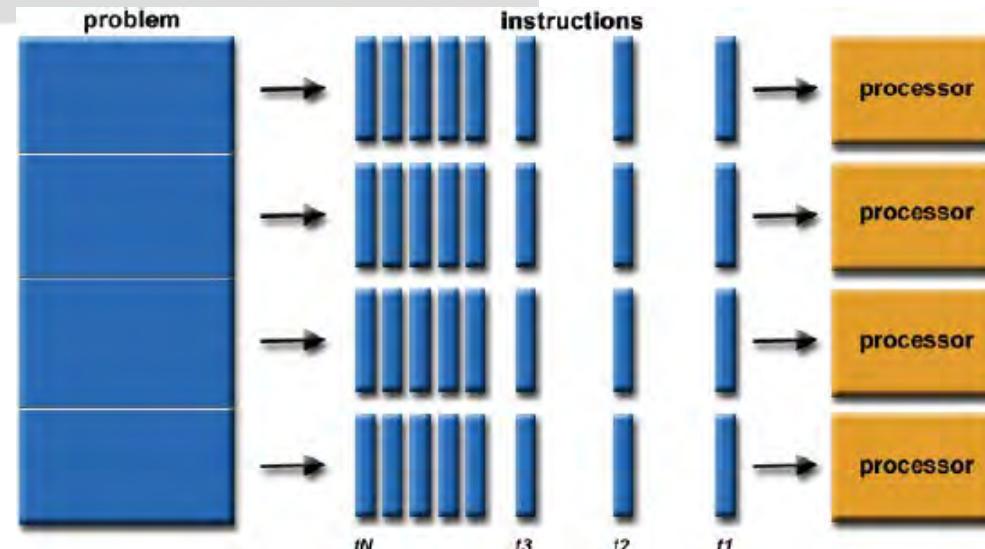
Need

- In distributed computing,
 - ✓ Multiple computing resources are connected in network and computing tasks are distributed across these resources
 - ✓ Results into increase in speed and efficiency of system
 - ✓ Faster and more efficient than traditional methods of computing
 - ✓ More suitable to process huge amounts of data in limited time



Parallel Computing

- In Parallel Computing,
 - ✓ Additional computational resources are added to the same system to improve the processing capability of system
 - ✓ Helps in dividing complex computations into subtasks, which can be handled individually by processing units that are running in parallel
 - ✓ Multiple computing systems are running parallel
 - ✓ Concept is processing capability will increase with the increase in the level of parallelism



Techniques for High Data Processing

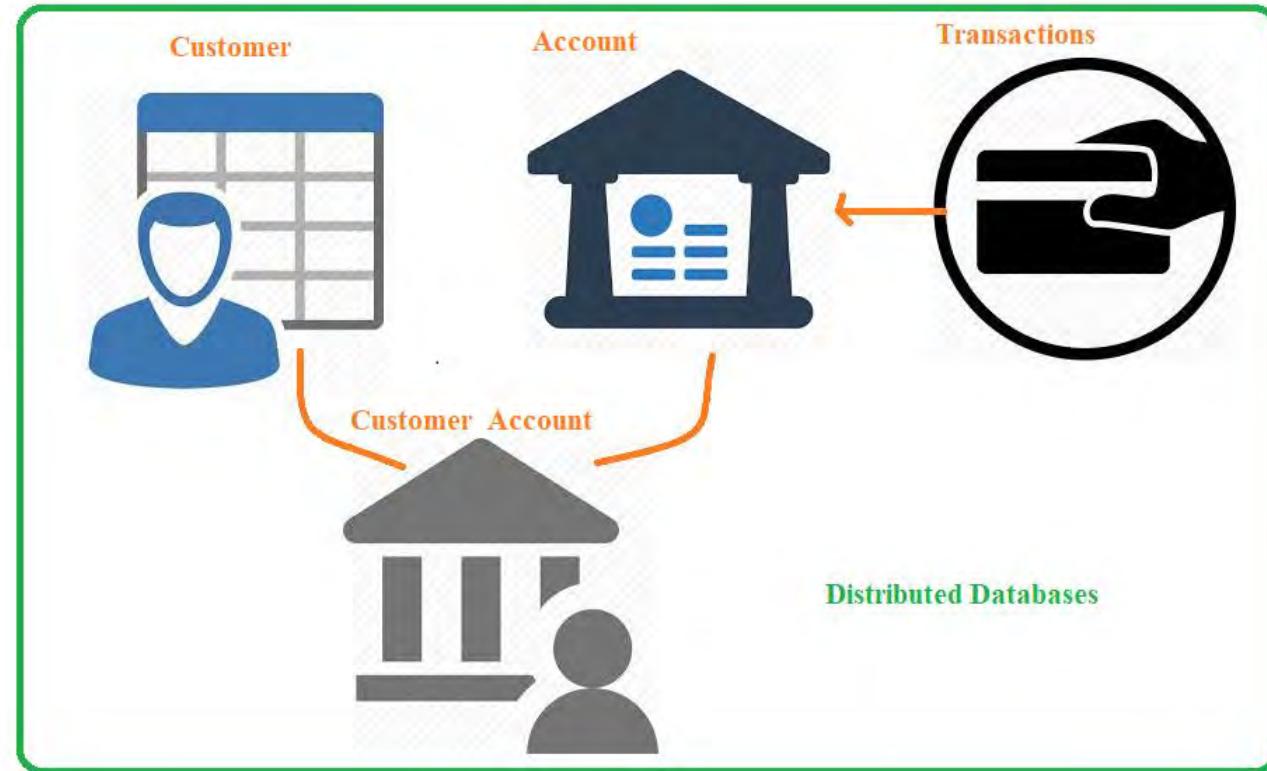
Method	Description	Usage
Cluster or Grid Computing	<ul style="list-style-type: none">Based on a connection of multiple servers in a networkServers share the workload among themSystems can be homogeneous or heterogeneous	<ul style="list-style-type: none">Used in HadoopCluster can be created by hardware components that were acquired long back
Massively Parallel Processing (MPP)	<ul style="list-style-type: none">Single machine working as gridCapable of handling activities of storage, memory and computing	<ul style="list-style-type: none">Used in Data warehousesEMC GreenplumParAccel
High-Performance Computing (HPC)	<ul style="list-style-type: none">Known to offer high performance and scalability by using IMCSuitable for processing floating-point data at high speeds	<ul style="list-style-type: none">Used to develop specialty and custom applications for research where results is more valuable than cost

Comparing Parallel and Distributed Systems

Distributed System	Parallel System
<ul style="list-style-type: none">Independent, autonomous system connected in a network accomplishing specific tasks	<ul style="list-style-type: none">Computer system with several processing units attached to it
<ul style="list-style-type: none">Coordination is possible between connected computers with own memory and CPU	<ul style="list-style-type: none">A common shared memory can be directly accessed by every processing unit in a network
<ul style="list-style-type: none">Loose coupling of computers connected in network, providing access to data and remotely located resources	<ul style="list-style-type: none">Tight coupling of processing resources that are used for solving single, complex problem

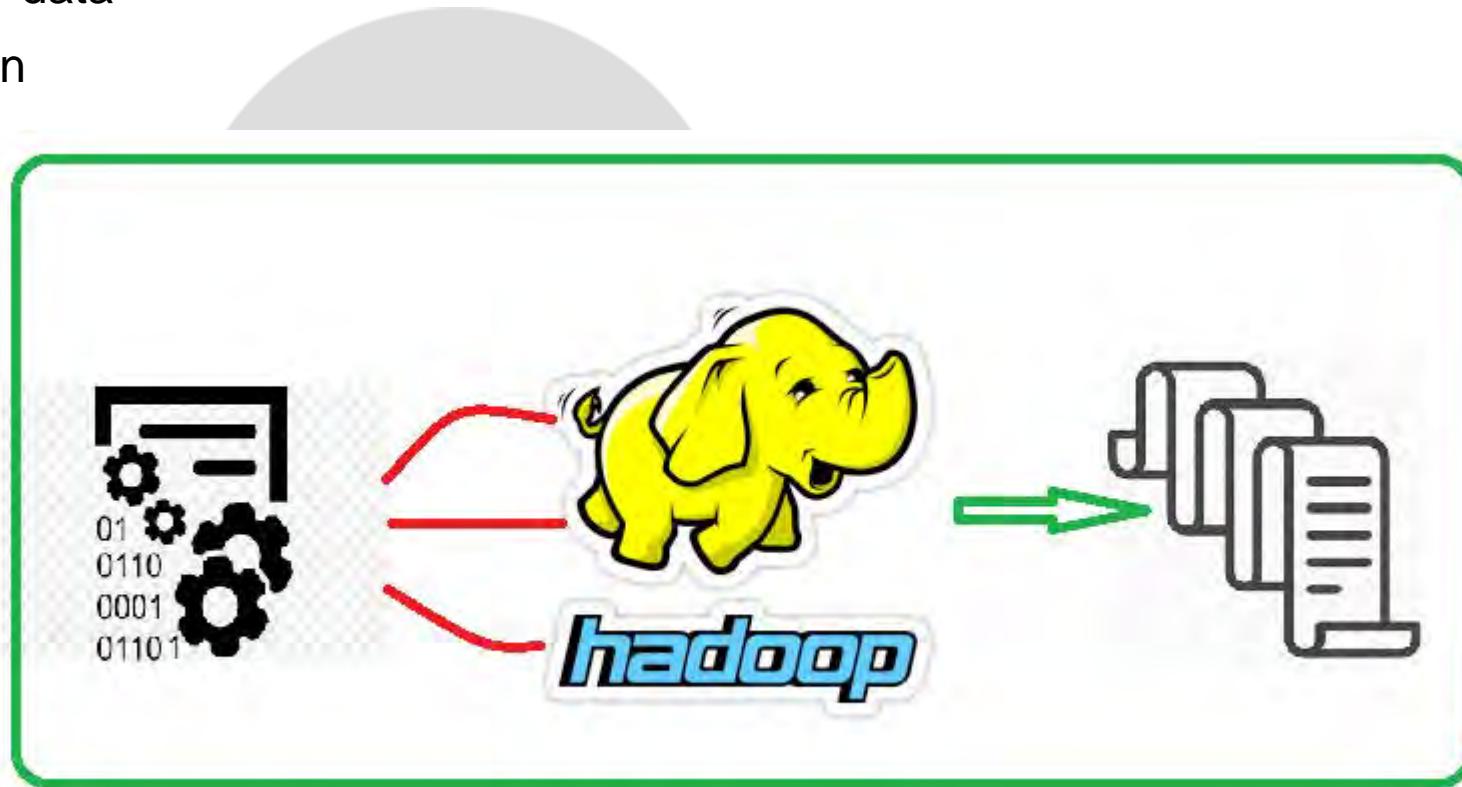
Data Models – Distributed Databases

- Deal with tables and relations
- Must have a schema for data
- Implements data partitioning



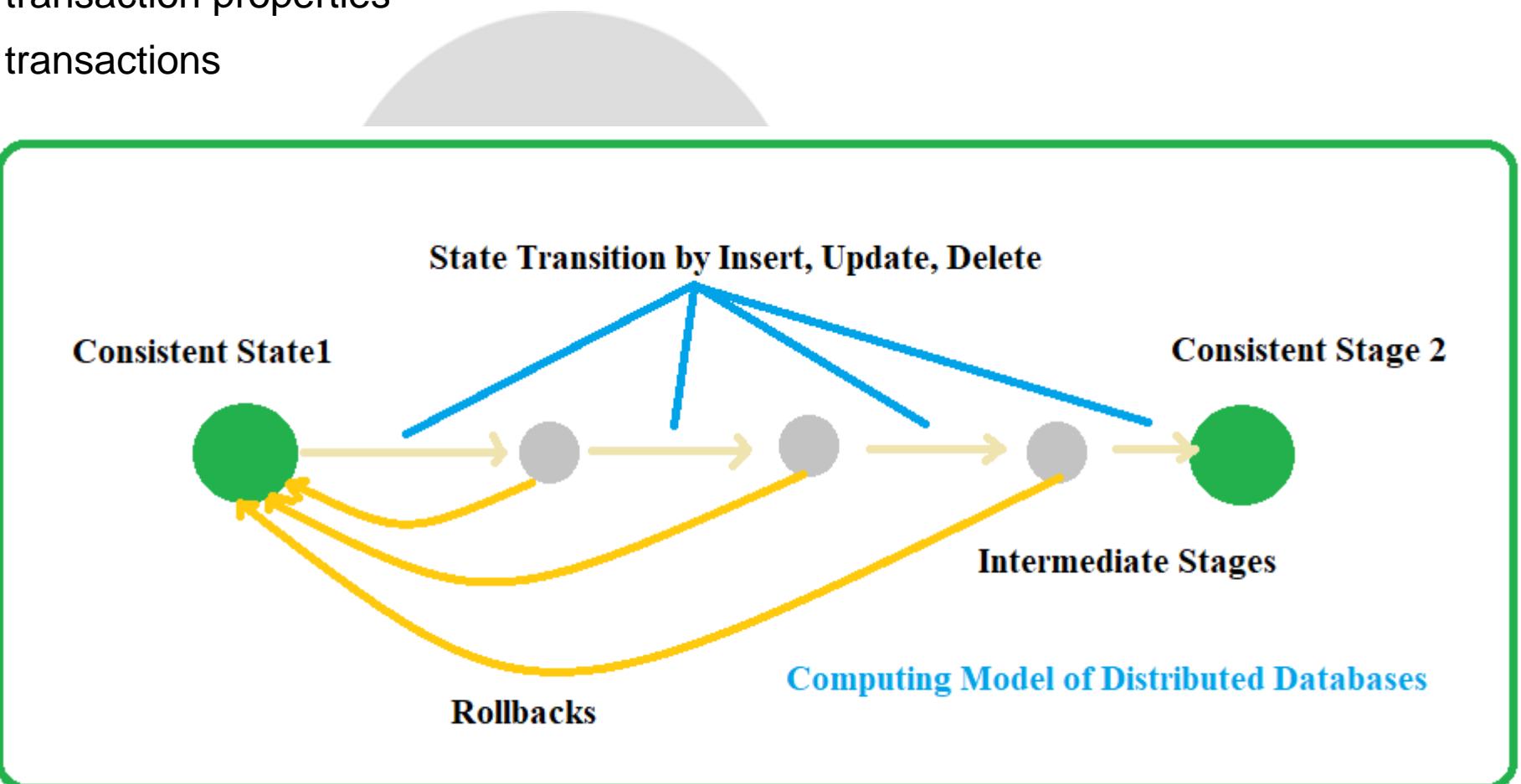
Data Models – Hadoop

- Deals with flat files in any format
- Operates on no schema for data
- Divides files automatically in blocks



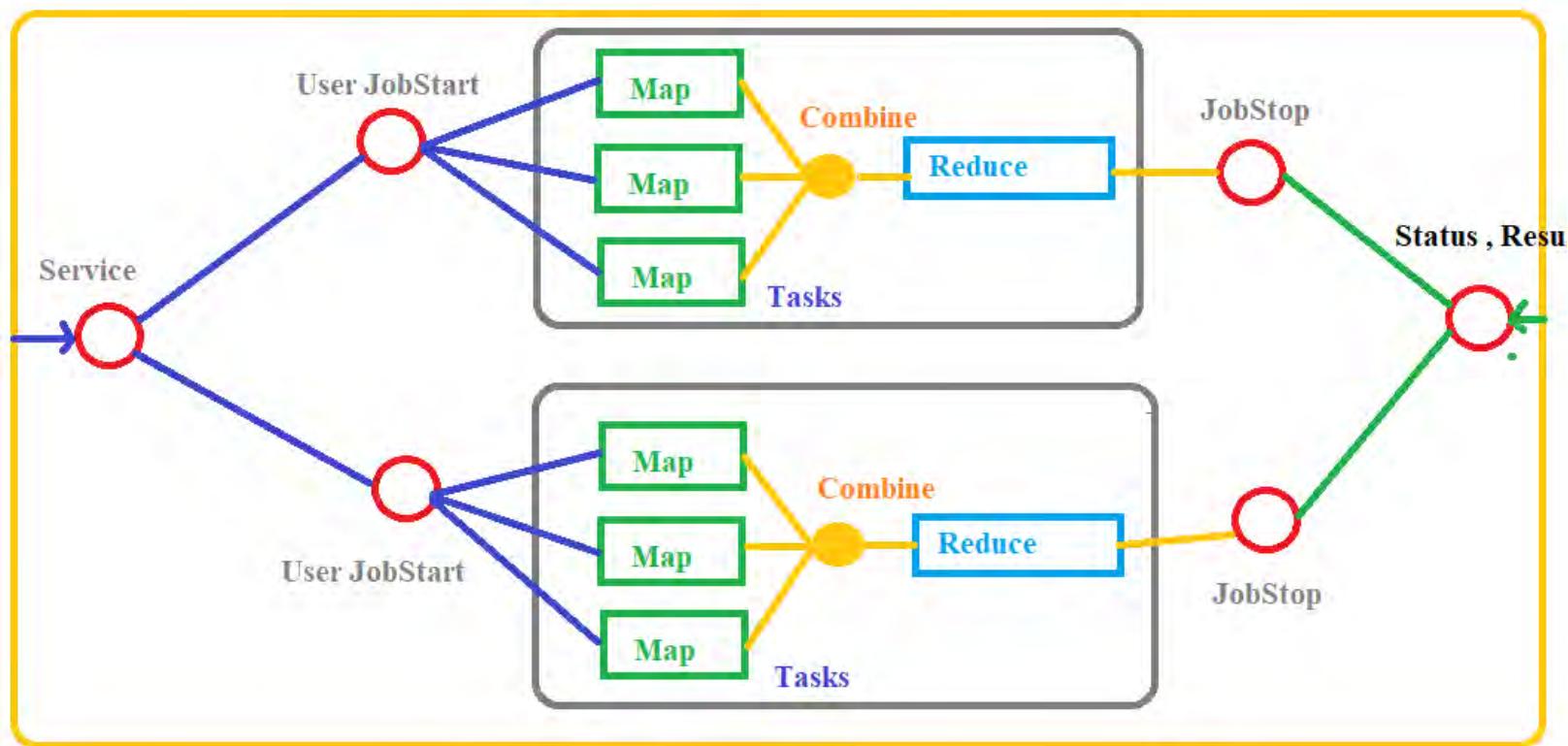
Computing model – Distributed Databases

- Generate notions of transaction
- Implement ACID transaction properties
- Allow distributed transactions



Computing model - Hadoop

- Generates notions of a job divided into tasks
- Implements MapReduce computing model
- Considers every task either as a Map or a Reduce





Thank You!

In our next session: Apache Hadoop for Big Data



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to Hadoop

Pravin Y Pawar

Big Data Challenges

Volume, Variety and Velocity

- “How to store terabytes of mounting data?”
 - ✓ VOLUME
 - “How to handle structured, semi-structured and unstructured data?”
 - ✓ VARIETY
 - “How to manage the data that is getting generated at very fast speed?”
 - ✓ VELOCITY

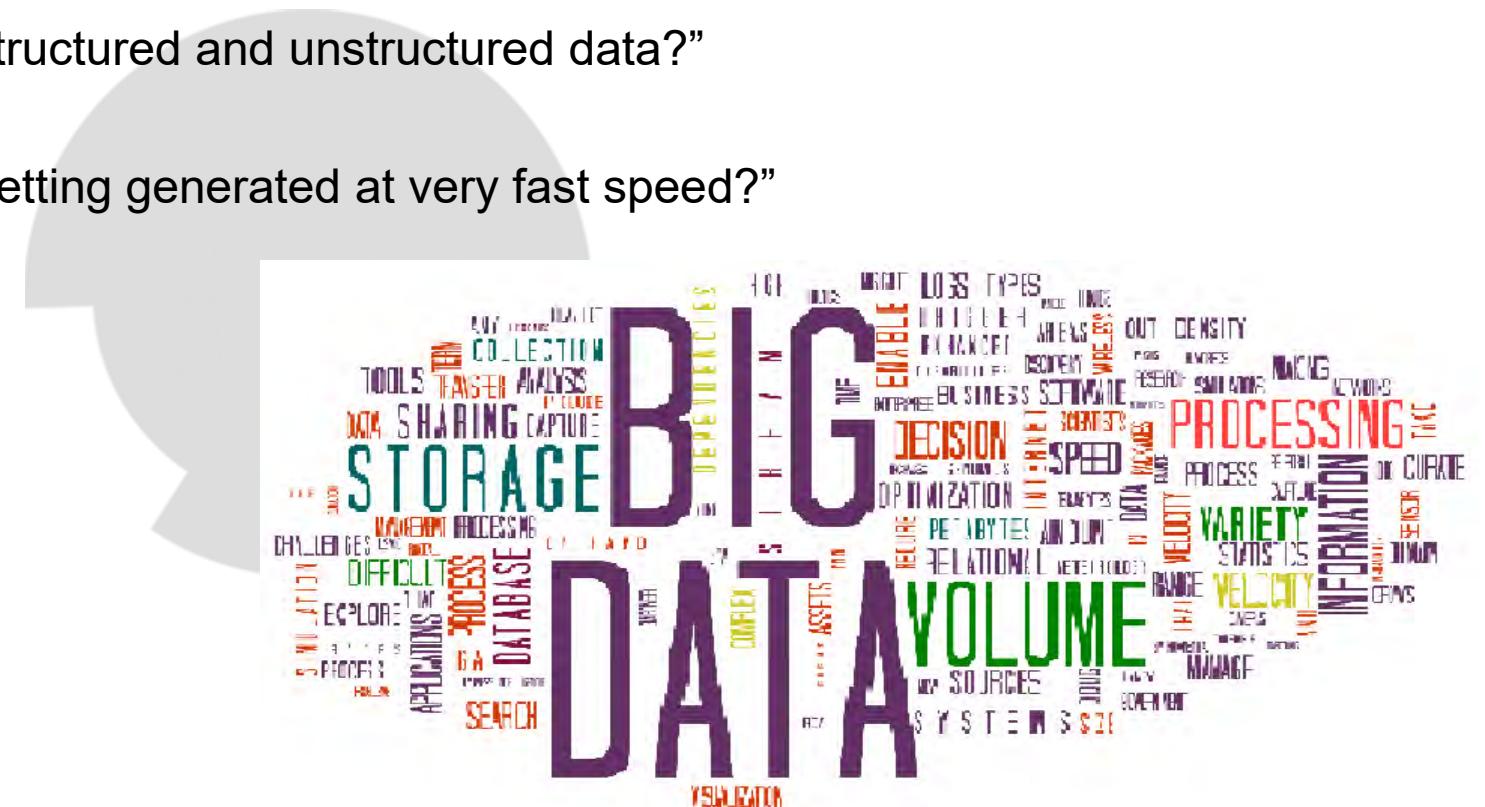


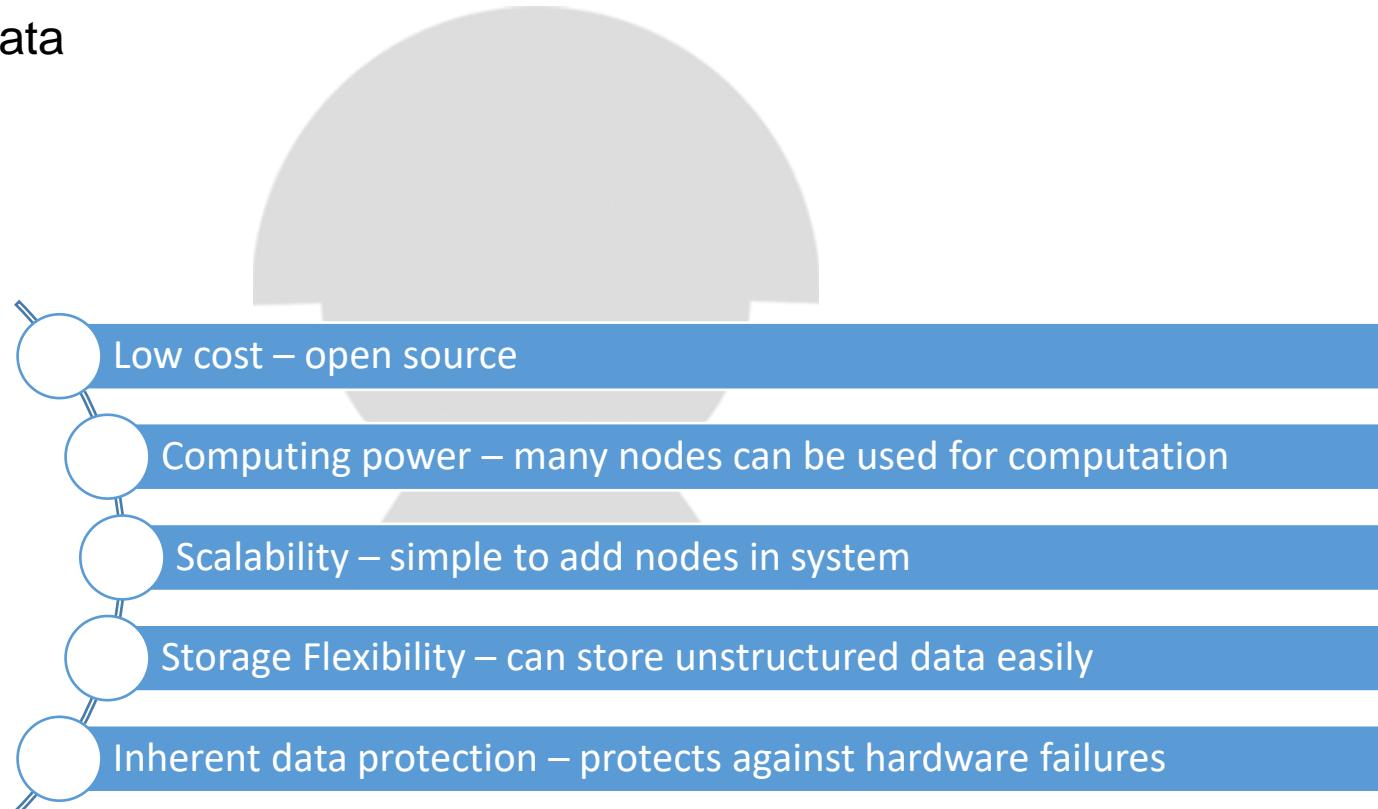
Image Source : [amcham](#)

Why Hadoop?

Key consideration

- Hadoop can handle
 - ✓ Massive amount of data
 - ✓ Different kinds of data
 - ✓ In fast manner

- Advantages



Distributed Computing Challenges

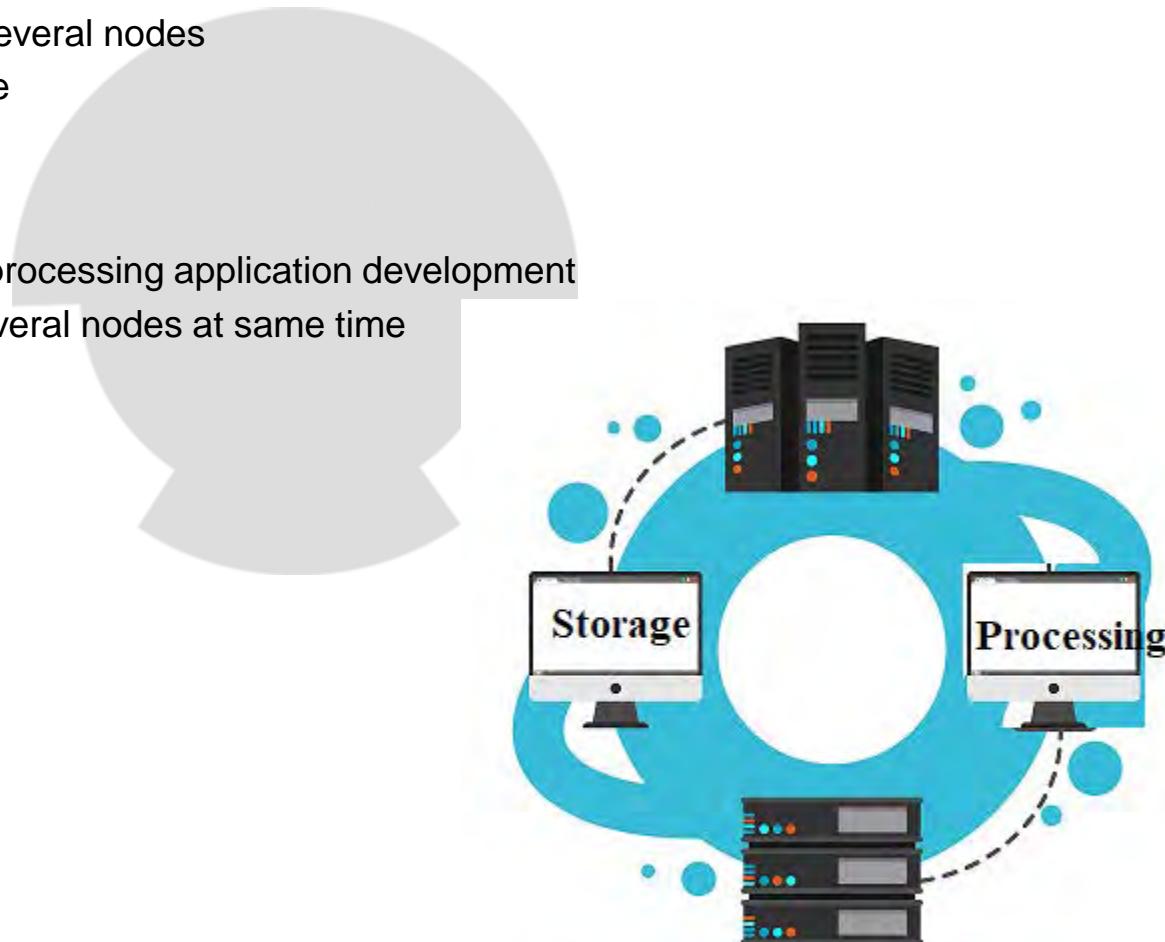
Problems and Solutions

- Storage of huge amount of data
 - ✓ More systems , more failures
 - ✓ How to retrieve the data stored on the failed node?
 - ✓ Hadoop solves this by Replication Factor (RF)
 - ✓ Number of data copies of a given data item / data block stored across the network
- Processing the huge amount of data
 - ✓ Data is spread across systems, how to process it in quick manner?
 - ✓ Challenge is to integrate data from different machines before processing
 - ✓ Hadoop solves this by MapReduce Programming
 - ✓ Programming model to process huge amount of data at same time in quick manner

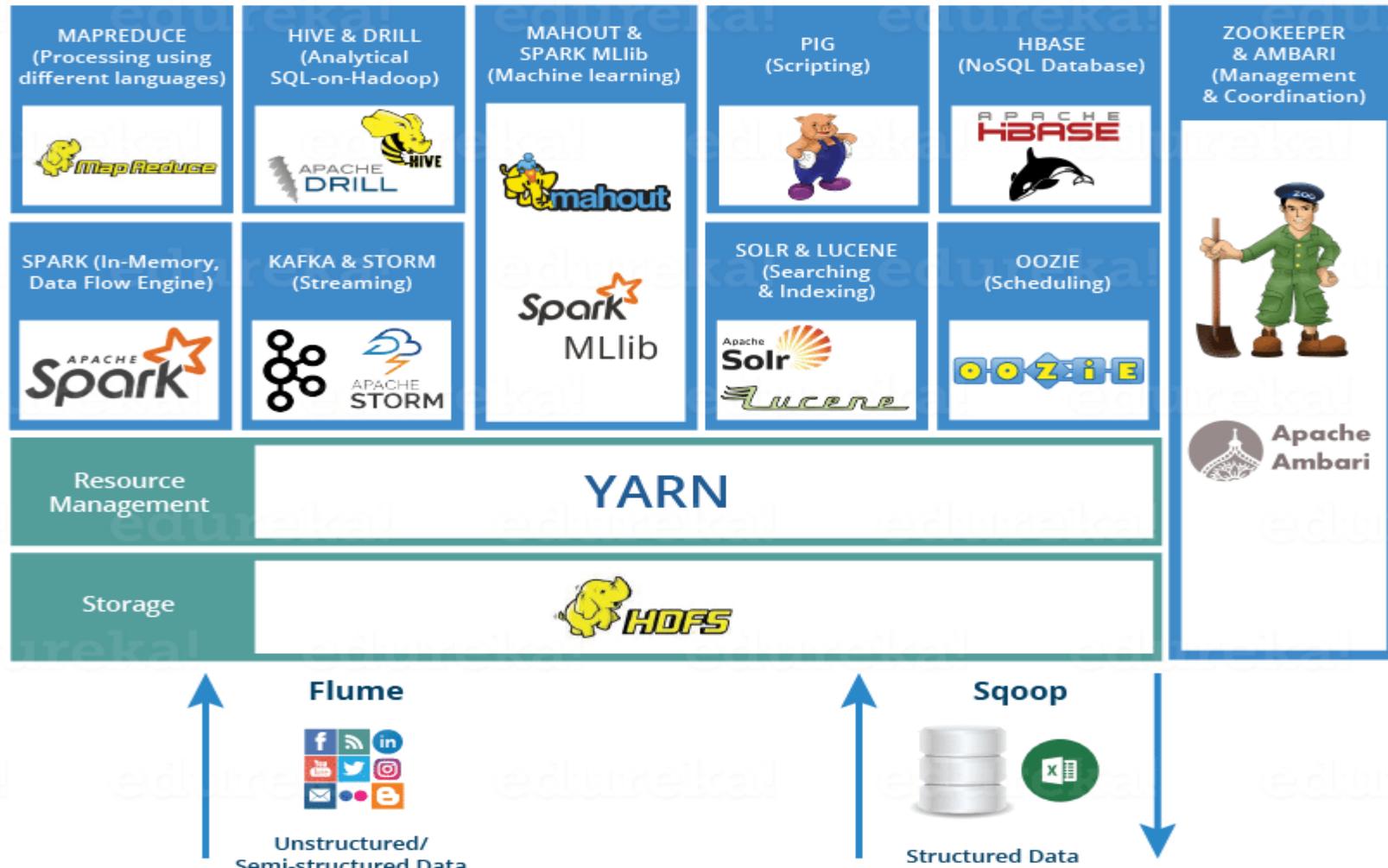
What is Hadoop?

Key Aspects

- Two Tasks
 - ✓ Massive Data Storage
 - ❑ Huge amount of data across several nodes
 - ❑ Uses low cost commodity storage
 - ✓ Faster Data Processing
 - ❑ Has everything needed for data processing application development
 - ❑ Computation done parallel on several nodes at same time



Hadoop Ecosystem



Hadoop High Level Architecture

High Level Architecture Of Hadoop

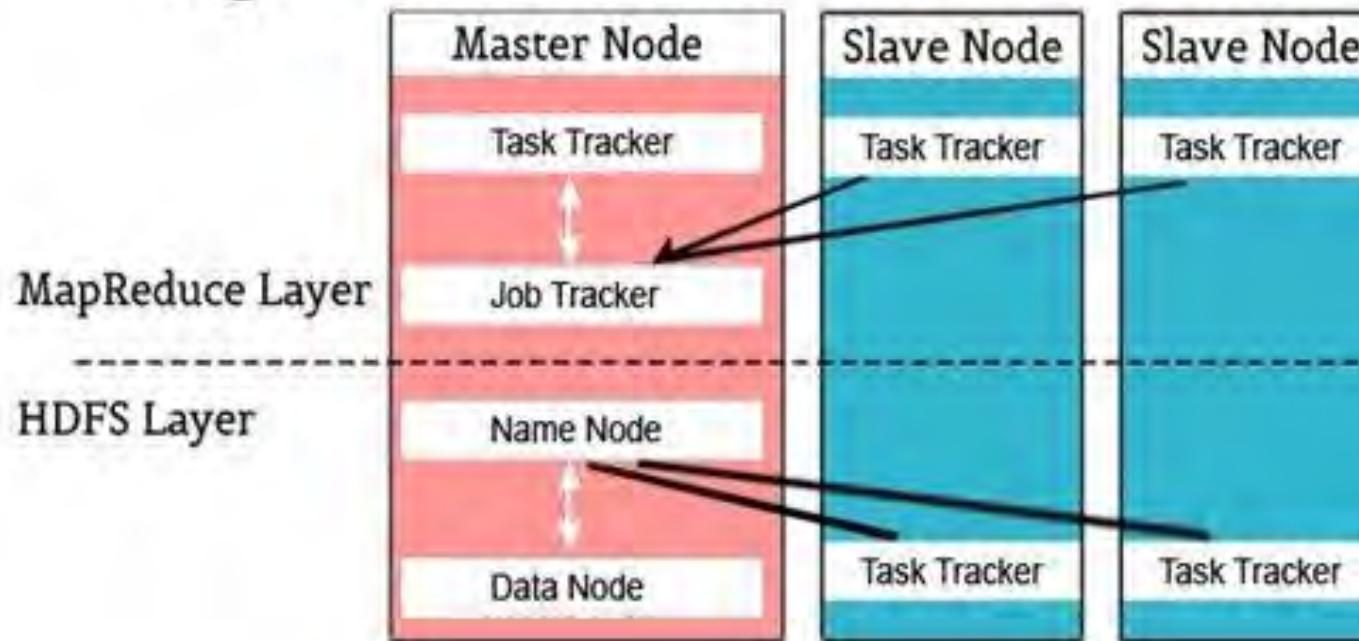


Image Source : [intellipaat](#)



Thank You!

In our next session: Cloud Computing for Big Data



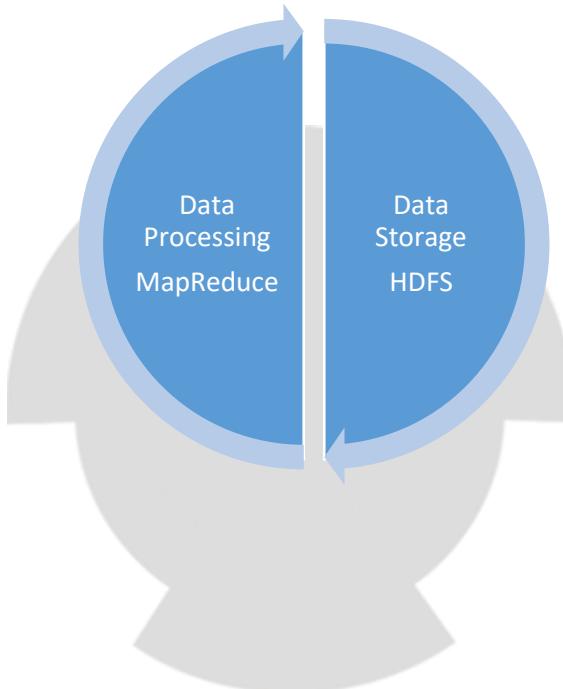
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Hadoop Architecture

Pravin Y Pawar

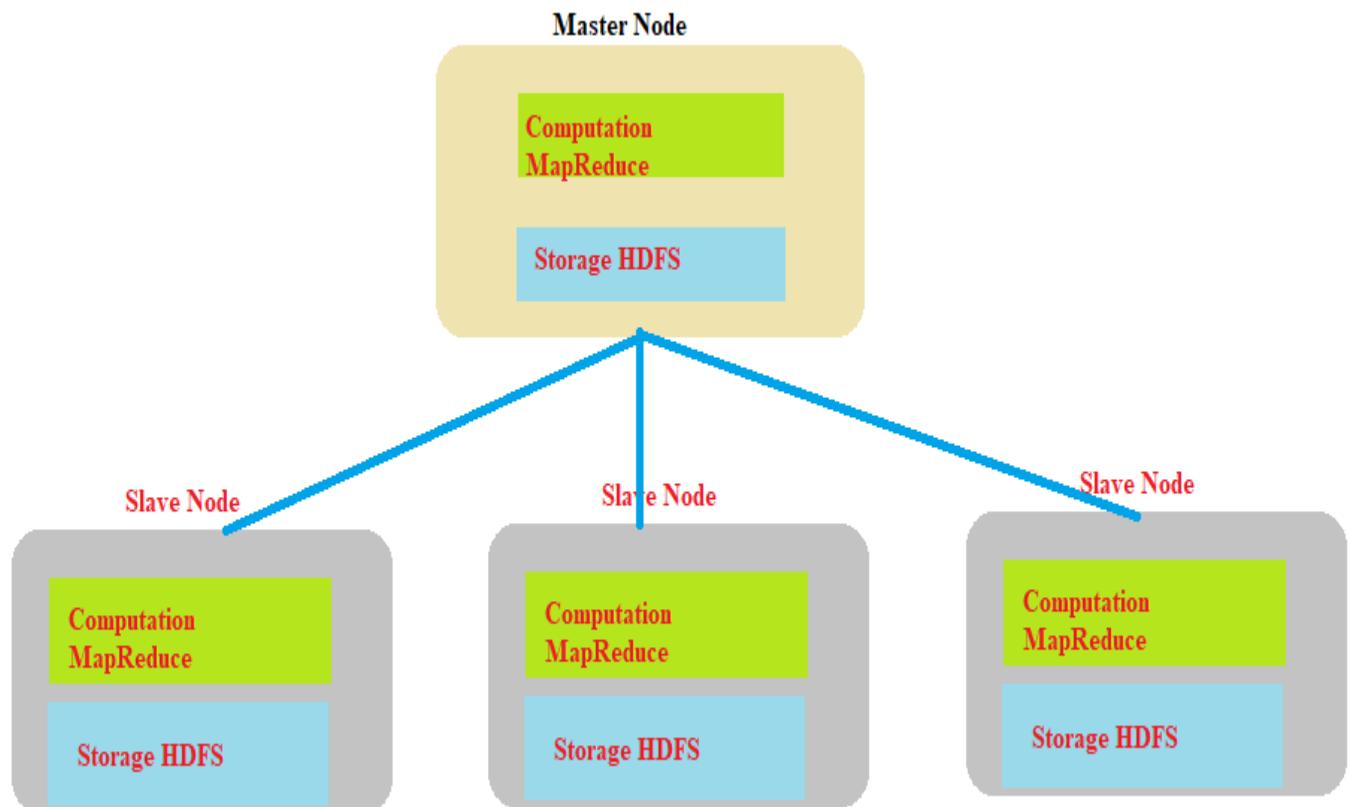
Conceptual Layer

- Conceptually divided into
 - ✓ Data Storage Layer
 - ✓ Data Processing Layer



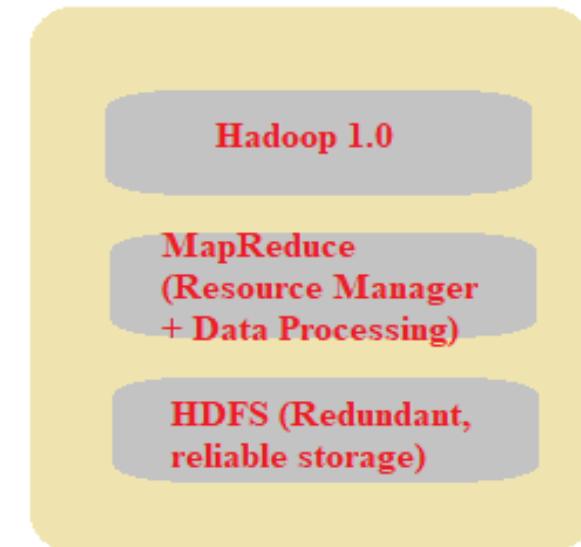
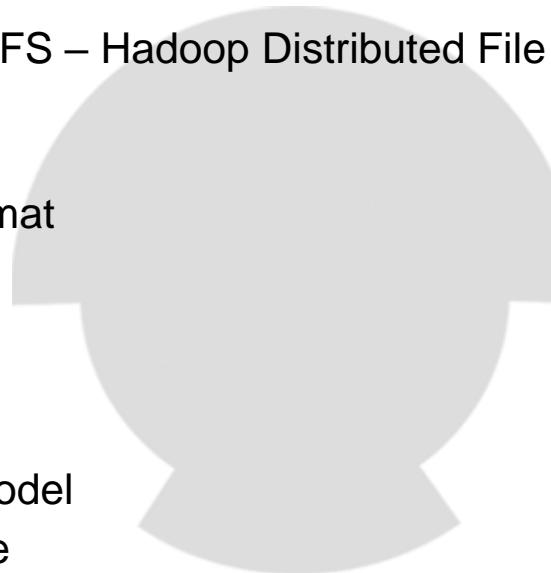
High Level Architecture

- Master Slave Architecture
- Master Node: NameNode
- Slave Node : DataNode
- Components of Master Node
 - ✓ Master HDFS
 - ✓ Responsible for partitioning the data storage across slave nodes
 - ✓ Keeps tracks of locations of data on DataNodes
 - ✓ Master MapReduce
 - ✓ Decides and schedules computation task on slave nodes



Hadoop Version 1.0

- Two main parts
- Data Storage Framework
 - ✓ General purpose file system – HDFS – Hadoop Distributed File System
 - ✓ Schema less
 - ✓ Simply stores data files in any format
 - ✓ Provides a lot of flexibility
- Data Processing Framework
 - ✓ Simple functional programming model
 - ✓ Formed by Google as MapReduce
 - ✓ Map – take set of key-value pairs and generate intermediate data which is also key-value pair
 - ✓ Reduce – act of the intermediate key-value pairs to output data



Hadoop High Level Architecture

High Level Architecture Of Hadoop

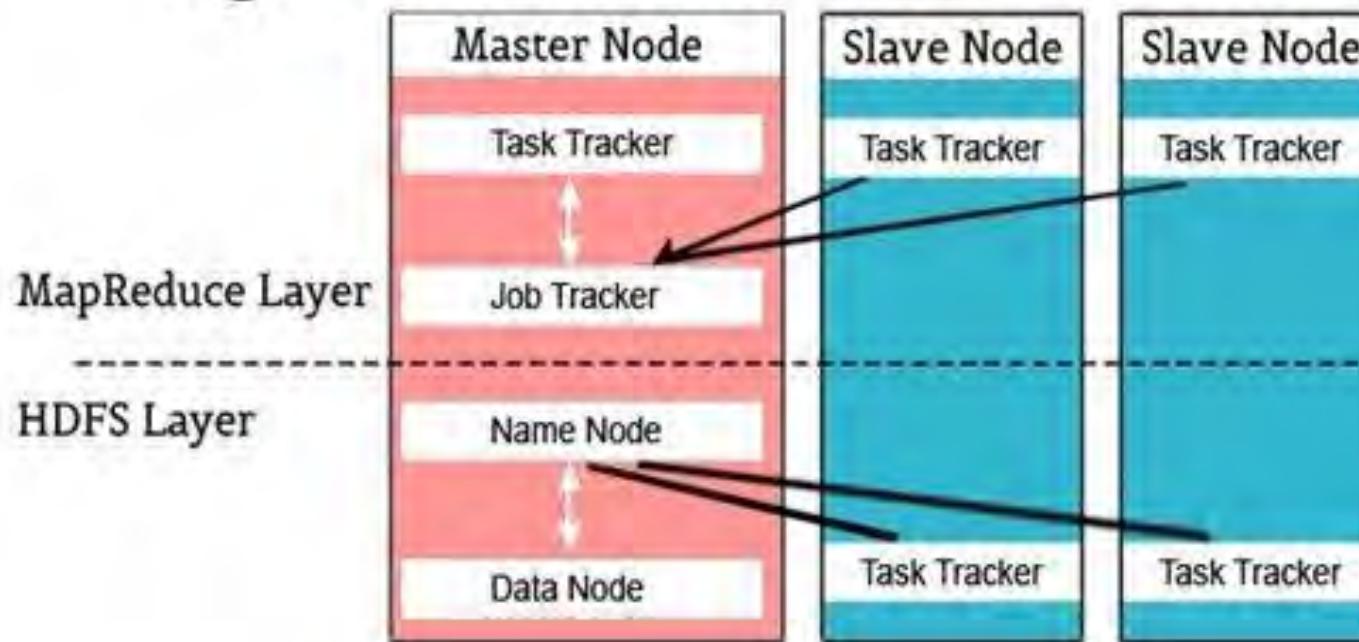


Image Source : [intellipaat](#)

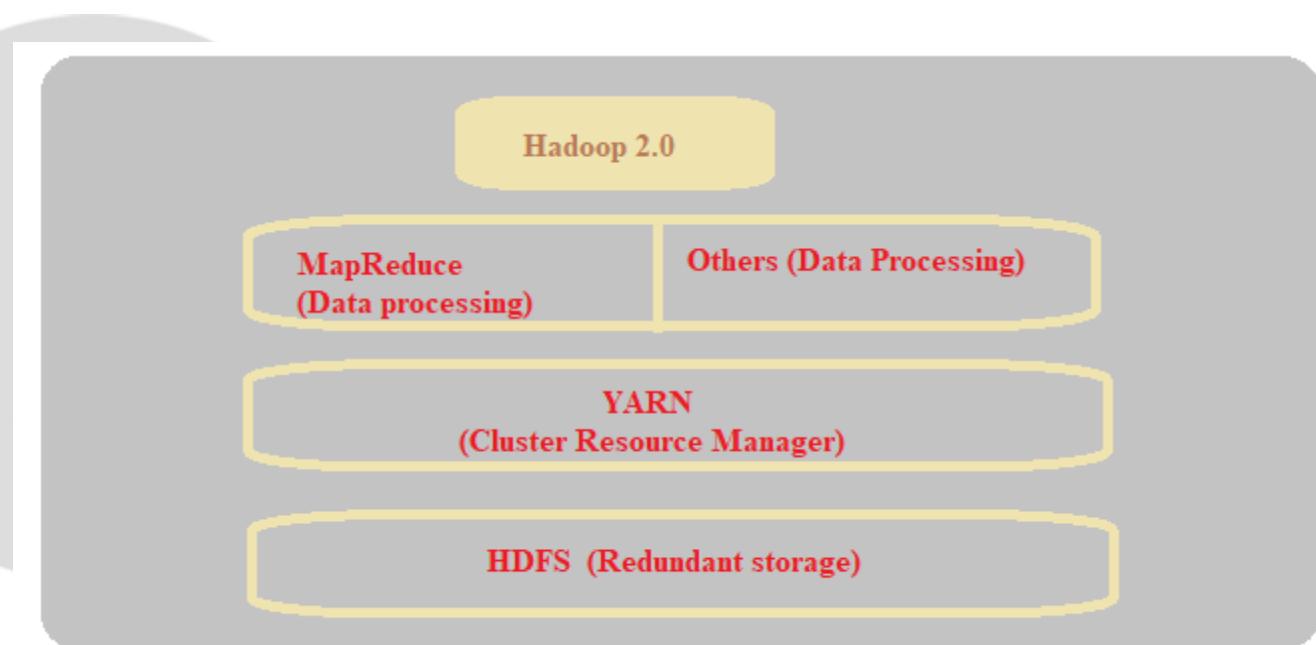
Hadoop Version 1.0

Limitations

- Requirement of MapReduce expertise along with proficiency in language like Java
- Only supports batch processing jobs like log analysis, data processing
- Tightly coupled with MapReduce
 - ✓ Only Two solutions
 - ✓ Needs to adapt the existing data processing program into MapReduce paradigm
 - ✓ Extract data from HDFS and execute the programs in other platforms
 - ✓ None is viable as results into inefficiencies by data movement!

Hadoop Version 2.0

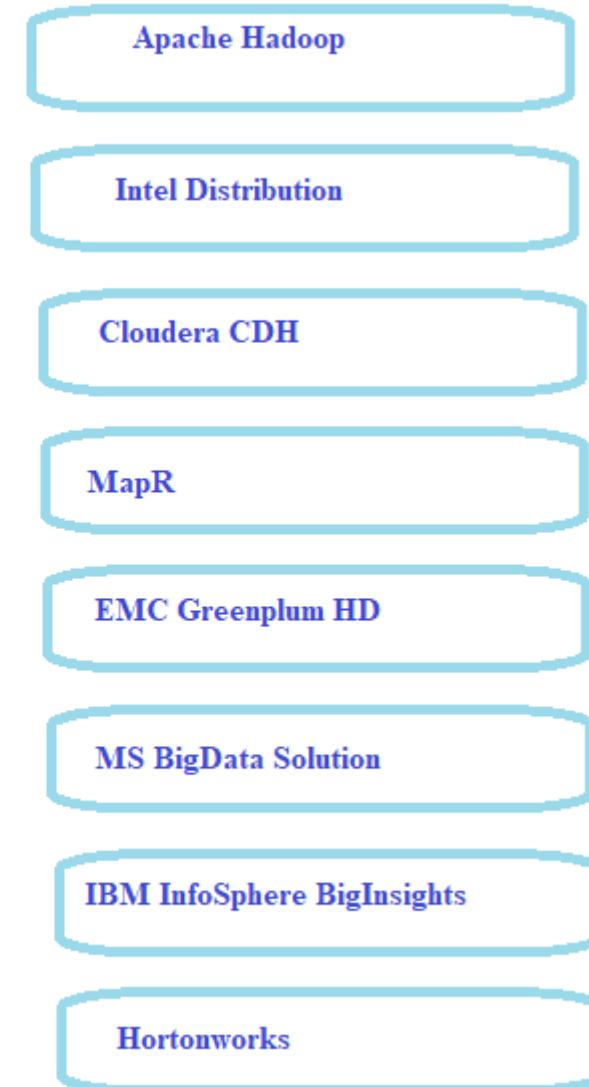
- New resource manager YARN – Yet Another Resource Navigator is added
- Any application capable of diving itself into parallel tasks is supported by YARN
- Takes care of allocations of subtasks of submitted applications
 - ✓ ApplicationMaster instead of JobTracker on Master node
 - ✓ NodeManager instead of TaskTracker on the Slave nodes
- Removed dependency on the MapReduce
- Supports real time workloads as well



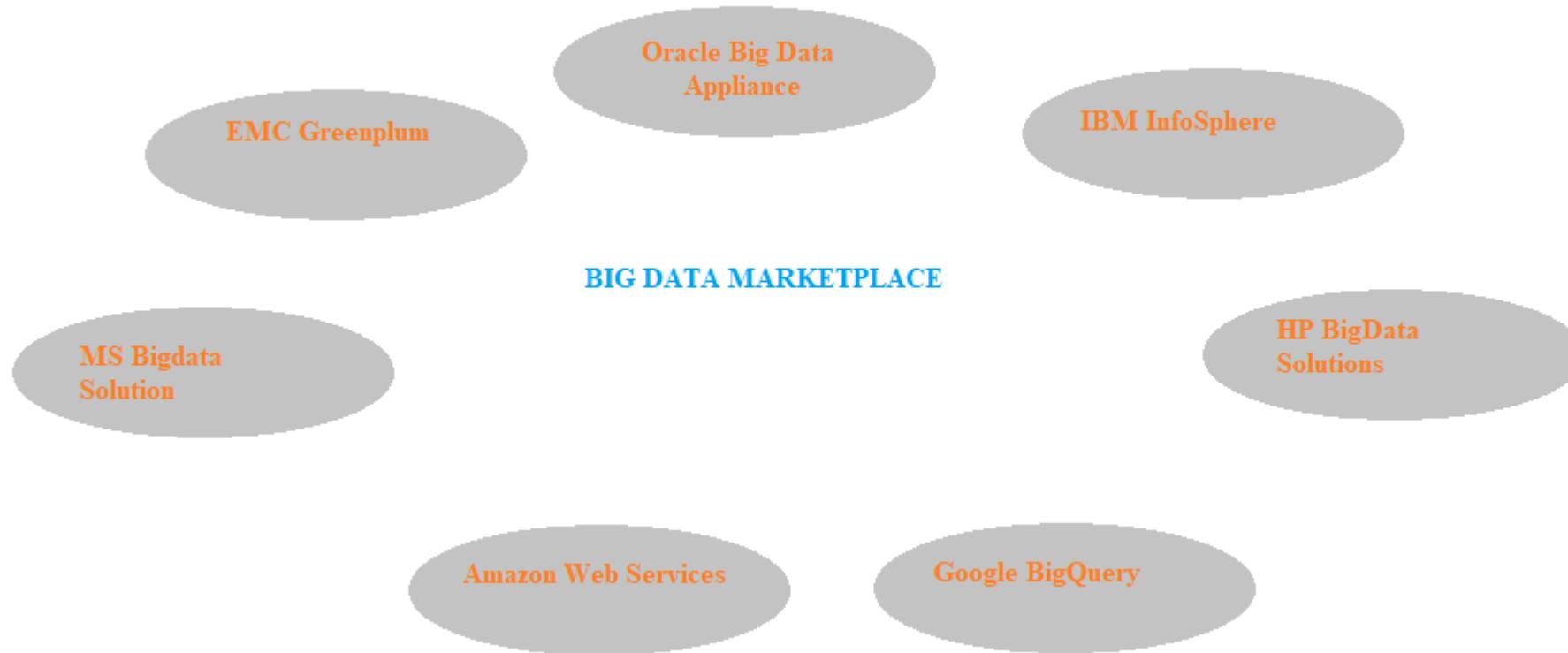
Hadoop Distributions

- Open source Apache project
- Core components :
 - ✓ Hadoop Common
 - ✓ Hadoop Distributed File System
 - ✓ Hadoop YARN
 - ✓ Hadoop MapReduce

Hadoop Distribution



Leading Vendors



Reference :

Big Data and Analytics by Acharya, Chellappan



Thank You!

In our next session: Hadoop Ecosystem



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

In-Memory Computing for Big Data

Pravin Y Pawar

Hadoop for Big Data

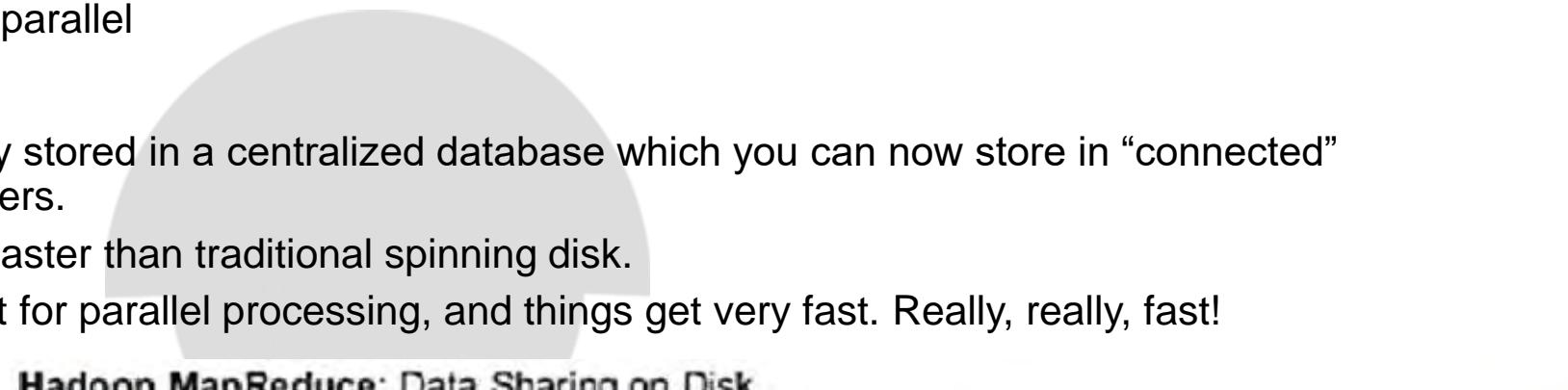
Issues with MapReduce on Hadoop

- Apache Hadoop
 - ✓ Revolutionized big data processing, enabling users to store and process huge amounts of data at very low costs.
 - ✓ An ideal platform to implement complex batch applications as diverse as
 - sifting through system logs
 - running ETL
 - computing web indexes
 - recommendation systems etc.
 - ✓ Its reliance on persistent storage to provide fault tolerance and its one-pass computation model make MapReduce a poor fit for
 - low-latency applications
 - iterative computations, such as machine learning and graph algorithms.

Adapted from : <https://databricks.com/blog/2013/11/21/putting-spark-to-use.html>

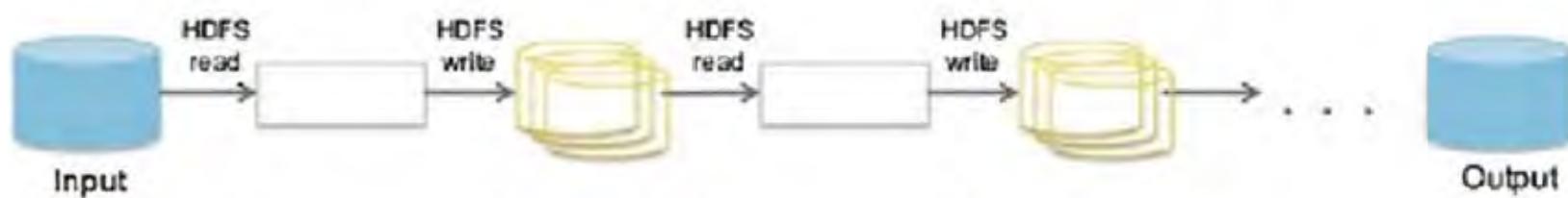
In-Memory Computing

- In-memory computing
 - ✓ means using a type of middleware software that allows one to store data in RAM, across a cluster of computers, and process it in parallel
- For example,
 - ✓ Operational datasets typically stored in a centralized database which you can now store in “connected” RAM across multiple computers.
 - ✓ RAM is roughly 5,000 times faster than traditional spinning disk.
 - ✓ Add to the mix native support for parallel processing, and things get very fast. Really, really, fast!



RAM storage and parallel distributed processing are two fundamental pillars of in-memory computing.

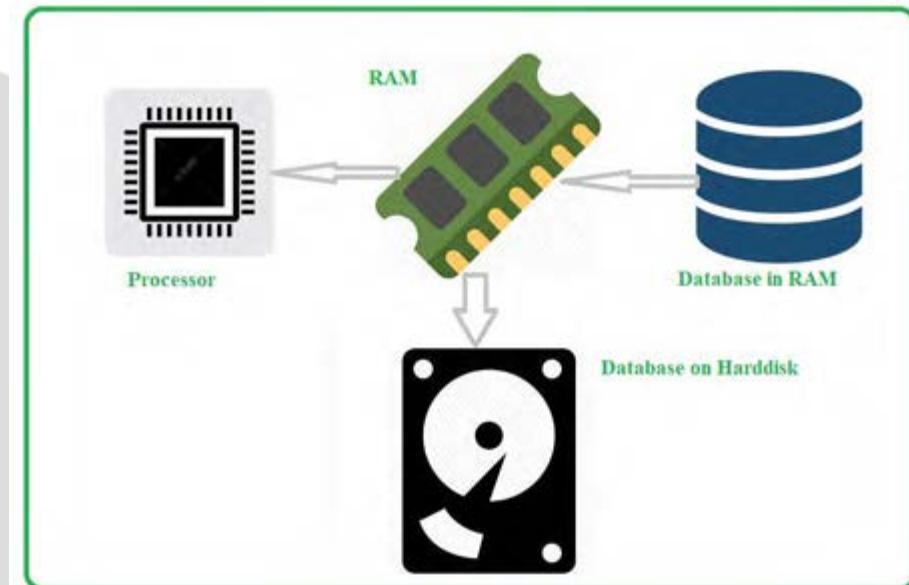
Hadoop MapReduce: Data Sharing on Disk



Spark: Speed up processing by using Memory instead of Disks

In-Memory Computing for Big Data

- RAM (or primary storage area)
 - ✓ is used for data processing
 - ✓ Helps to increase the computing speed
 - ✓ Reduction in prices made is affordable
- Data handling
 - ✓ Relational databases are used for structured data
 - ✓ Unstructured data is handled by NoSql databases
- Volume is addressed by IMC
- Diversity of big data managed by NoSQL databases

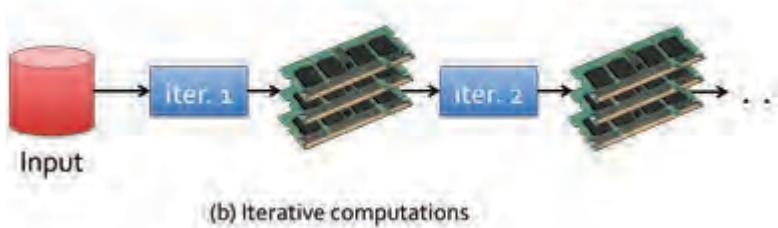
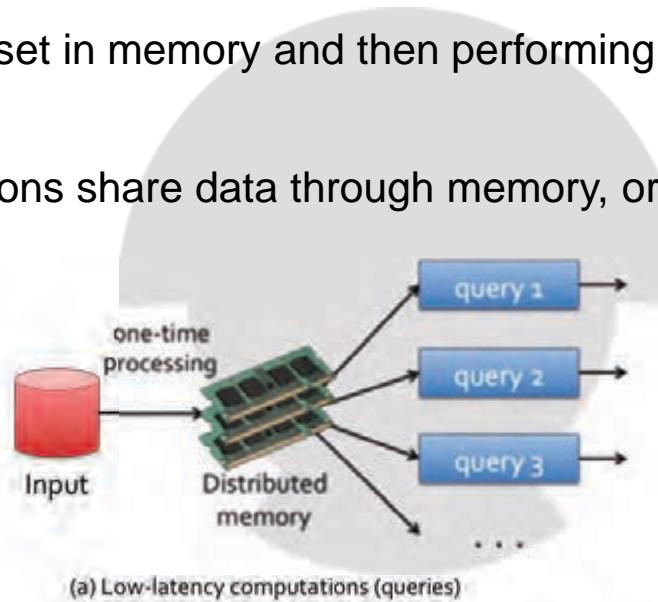


Fast and Easy Big Data Processing with Spark

- At its core, Spark provides a general programming model that enables developers to write application by composing arbitrary operators, such as
 - ✓ mappers
 - ✓ reducers
 - ✓ joins
 - ✓ group-bys
 - ✓ and filters
- This composition makes it easy to express a wide array of computations, including iterative machine learning, streaming, complex queries, and batch.
- Spark keeps track of the data that each of the operators produces, and enables applications to reliably store this data in memory.
 - ✓ This is the key to Spark's performance, as it allows applications to avoid costly disk accesses.

Apache Spark Applications

- Spark enables
- Low-latency computations
 - ✓ by caching the working dataset in memory and then performing computations at memory speeds
- Efficient iterative algorithm
 - ✓ by having subsequent iterations share data through memory, or repeatedly accessing the same dataset





Thank You!

In next lecture : Big Data Analytics Lifecycle



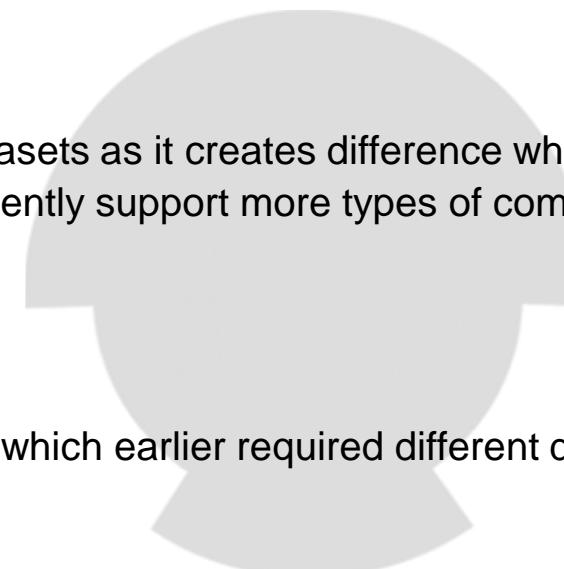
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Apache Spark

Pravin Y Pawar

Spark Overview

- Cluster computing platform
 - ✓ Designed to be fast and general purpose
- Speed
 - ✓ Is important in processing large datasets as it creates difference when data is being explored
 - ✓ Extends MapReduce model to efficiently support more types of computations
 - ✓ Runs computations in memory
- Generality
 - ✓ Covers a wide variety of workloads which earlier required different distributed systems
 - ✓ Including
 - ❖ Batch applications
 - ❖ Iterative algorithms
 - ❖ Interactive queries
 - ❖ Streaming
 - ✓ Easy and inexpensive to combine different processing types in data pipelines



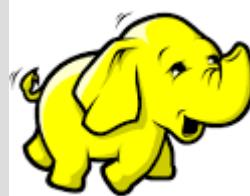
Spark Overview (2)

- Spark handles highly accessible, simple APIs in

- ✓ Java
- ✓ Python
- ✓ Scala
- ✓ SQL



- Integrates easily with other big data tools like
 - ✓ Hadoop (HDFS) and other ecosystem tools
 - ✓ Kafka
 - ✓ AWS



Unified Stack

Spark SQL
structured data

Spark Streaming
real-time

MLib
machine
learning

GraphX
graph
processing

Spark Core

Standalone Scheduler

YARN

Mesos

Spark Core

- Contains the basic functionality of Spark, including components for
 - ✓ task scheduling
 - ✓ memory management
 - ✓ fault recovery
 - ✓ interacting with storage systems etc.
- Home to the API that defines resilient distributed datasets (RDDs), which are Spark's main programming abstraction
- Provides many APIs for building and manipulating these collections



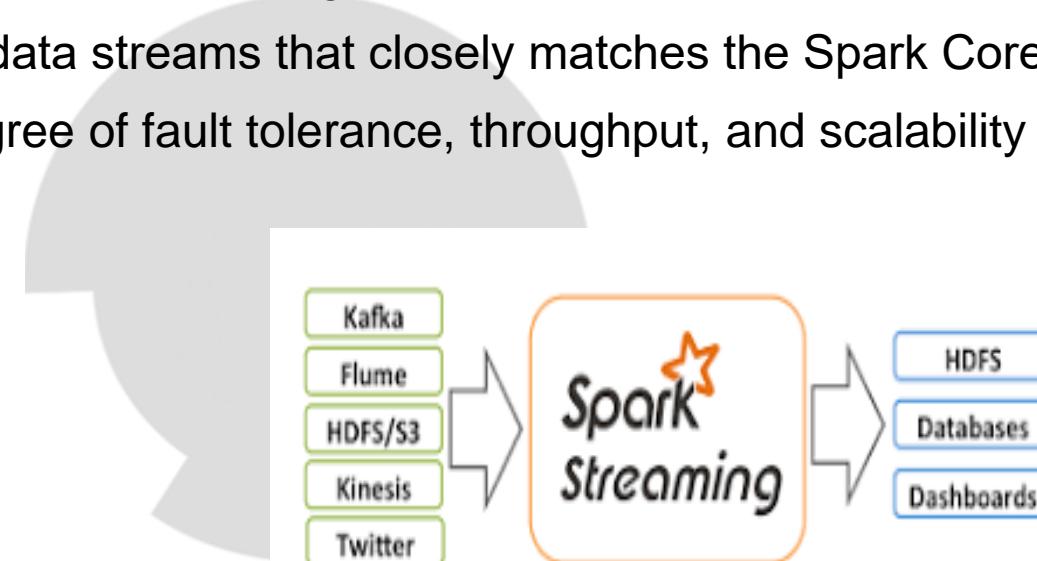
Spark SQL

- Spark's package for working with structured data
- Allows querying data via SQL as well as the Apache Hive variant of SQL
 - ✓ called the Hive Query Language (HQL)
- Allows developers to intermix SQL queries with the programmatic data manipulations
 - supported by RDDs in
 - ✓ Python
 - ✓ Java
 - ✓ Scala
 - all within a single application
- Tight integration with the rich computing environment provided by Spark makes Spark SQL unlike any other open source data warehouse tool



Spark Streaming

- Spark component that enables processing of live streams of data
- Examples of data streams include web servers log files
- Provides an API for manipulating data streams that closely matches the Spark Core's RDD API
- Designed to provide the same degree of fault tolerance, throughput, and scalability as Spark Core



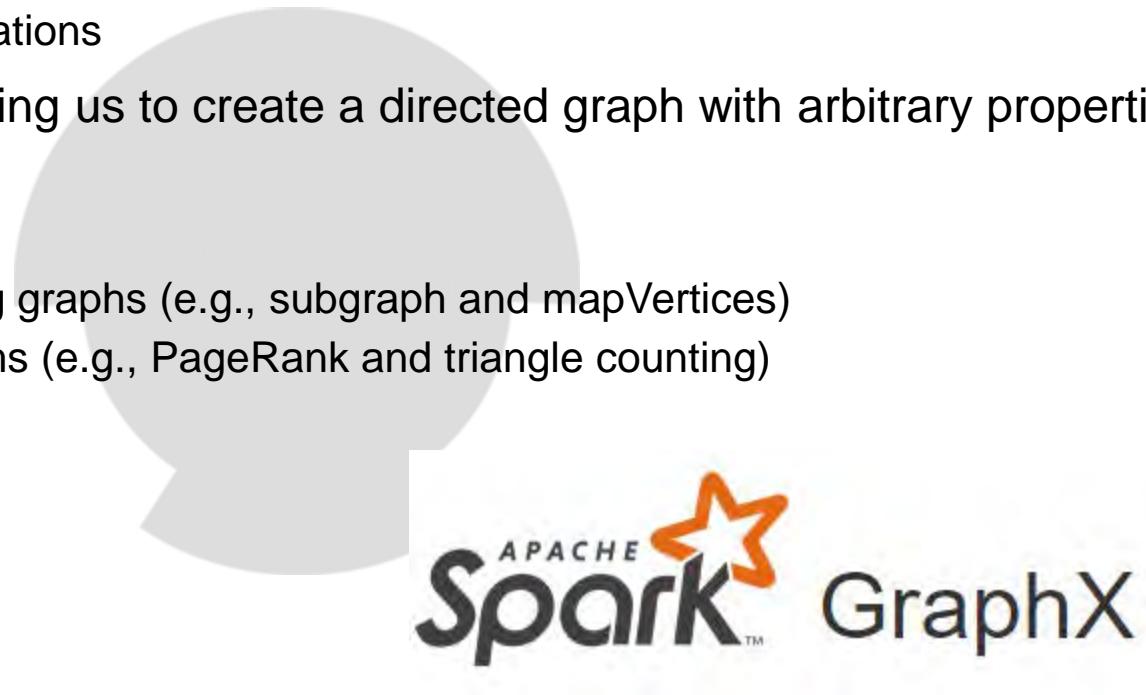
MLlib

- Built in library containing common machine learning (ML) functionality
- Provides multiple types of machine learning algorithms, including
 - ✓ Classification
 - ✓ Regression
 - ✓ Clustering
 - ✓ Collaborative filtering
 - ✓ Supporting functionality such as data import and model evaluation
- Provides some lower-level ML primitives, including a generic gradient descent optimization algorithm
- All of these methods are designed to scale out across a cluster



GraphX

- Library for
 - ✓ manipulating graphs (e.g. a social network's relations graph)
 - ✓ performing graph-parallel computations
- Extends the Spark RDD API, allowing us to create a directed graph with arbitrary properties attached to each vertex and edge
- Provides
 - ✓ various operators for manipulating graphs (e.g., subgraph and mapVertices)
 - ✓ library of common graph algorithms (e.g., PageRank and triangle counting)



Cluster Managers

- Spark is designed to efficiently scale up from one to many thousands of compute nodes
- Can run over a variety of cluster managers, including
 - ✓ Hadoop YARN
 - ✓ Apache Mesos
 - ✓ Simple cluster manager included in Spark itself called the Standalone Scheduler
 - ✓ Kubernetes
- If you are just installing Spark on an empty set of machines
 - ✓ the Standalone Scheduler provides an easy way to get started
- If you already have a Hadoop YARN or Mesos cluster,
 - ✓ Spark's support for these cluster managers allows your applications to also run on them



Reference :

Learning Spark By Karau, Konwinski



Thank You!

In our next session: Spark – Getting Started



BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Data Warehousing

M6: OLAP & Multidimensional Databases (MDDB)



6.1 Introduction to OLAP

Characteristics of Strategic Information

Integrated

Must have a single, enterprise-wide view

Data Integrity

Information must be accurate and must conform to business rules

Accessible

Easily accessible with intuitive access paths, and responsive for analysis

Credible

Every business factor must have one and only one value

Timely

Information must be available within the stipulated time frame

What is OLAP?

OLAP is a category of software technology that enables analysts, managers, and executives to gain insight into data through fast, consistent, interactive access to a wide variety of possible views of information that has been transformed from raw data to reflect the real dimensionality of the enterprise as understood by the user.

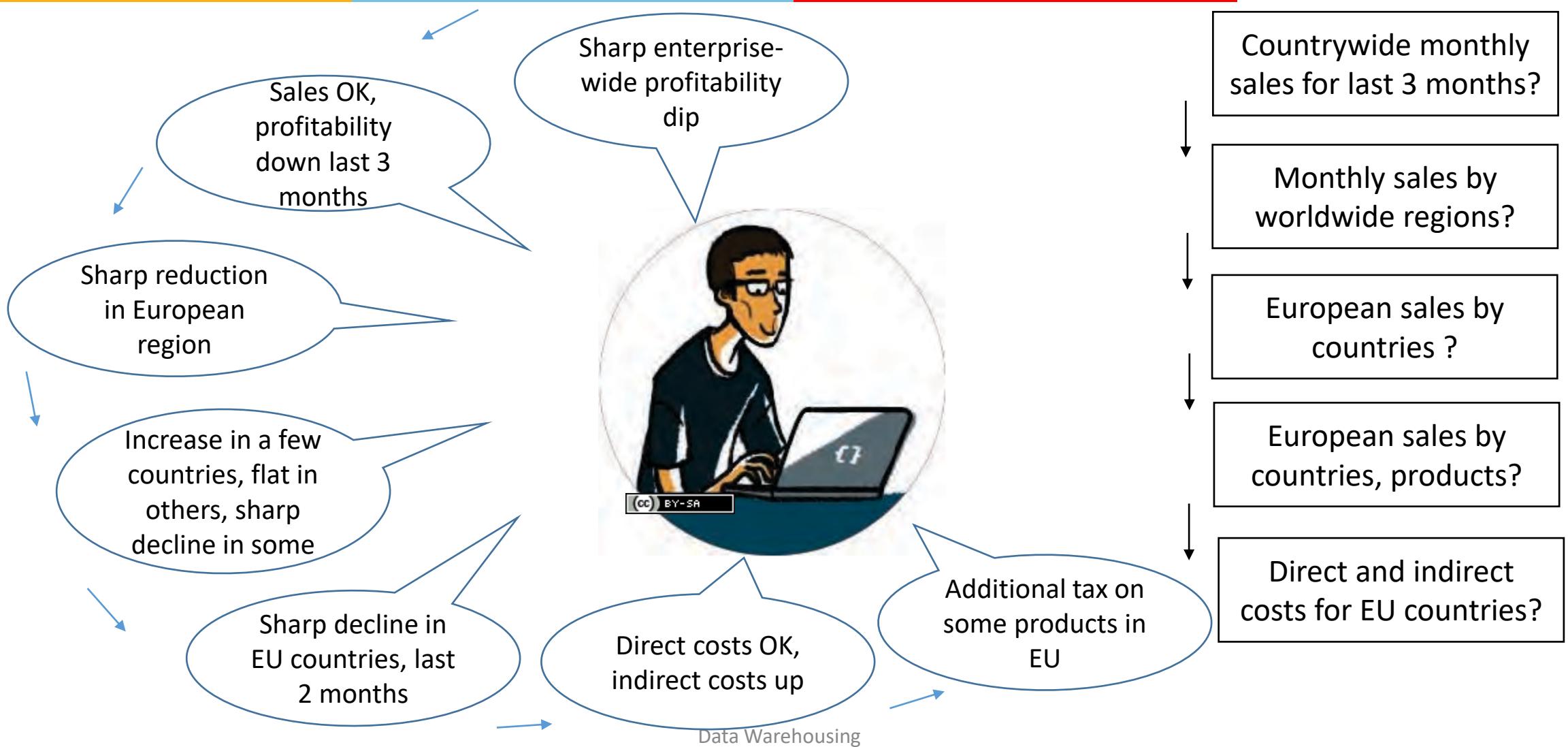
Codd's Rules for OLAP

1. **Multidimensional Conceptual View.** Provide a multidimensional data model that is intuitively analytical and easy to use. Business users' view of an enterprise is multidimensional in nature. Therefore, a multidimensional data model conforms to how the users perceive business problems.
2. **Transparency** Make the technology, underlying data repository, computing architecture, and the diverse nature of source data totally transparent to users. Such transparency, supporting a true open system approach, helps to enhance the efficiency and productivity of the users through front-end tools that are familiar to them.
3. **Accessibility** Provide access only to the data that is actually needed to perform the specific analysis, presenting a single, coherent, and consistent view to the users. The OLAP system must map its own logical schema to the heterogeneous physical data stores and perform any necessary transformations.
4. **Consistent Reporting Performance** Ensure that the users do not experience any significant degradation in reporting performance as the number of dimensions or the size of the database increases. Users must perceive consistent run time, response time, or machine utilization every time a given query is run.
5. **Client/Server Architecture** Conform the system to the principles of client/server architecture for optimum performance, flexibility, adaptability, and interoperability. Make the server component sufficiently intelligent to enable various clients to be attached with a minimum of effort and integration programming.
6. **Generic Dimensionality** Ensure that every data dimension is equivalent in both structure and operational capabilities. Have one logical structure for all dimensions. The basic data structure or the access techniques must not be biased toward any single data dimension.

Codd's Rules for OLAP

-
- 7. **Dynamic Sparse Matrix Handling** Adapt the physical schema to the specific analytical model being created and loaded that optimizes sparse matrix handling. When encountering a sparse matrix, the system must be able to dynamically deduce the distribution of the data and adjust the storage and access to achieve and maintain consistent level of performance.
 - 8. **Multiuser Support** Provide support for end users to work concurrently with either the same analytical model or to create different models from the same data. In short, provide concurrent data access, data integrity, and access security.
 - 9. **Unrestricted Cross-dimensional Operations** Provide ability for the system to recognize dimensional hierarchies and automatically perform roll-up and drill-down operations within a dimension or across dimensions. Have the interface language allow calculations and data manipulations across any number of data dimensions, without restricting any relations between data cells, regardless of the number of common data attributes each cell contains.
 - 10. **Intuitive Data Manipulation** Enable consolidation path reorientation (pivoting), drill-down and roll-up, and other manipulations to be accomplished intuitively and directly via point-and-click and drag-and-drop actions on the cells of the analytical model. Avoid the use of a menu or multiple trips to a user interface.
 - 11. **Flexible Reporting** Provide capabilities to the business user to arrange columns, rows, and cells in a manner that facilitates easy manipulation, analysis, and synthesis of information. Every dimension, including any subsets, must be able to be displayed with equal ease.
 - 12. **Unlimited Dimensions and Aggregation Levels** Accommodate at least fifteen, preferably twenty, data dimensions within a common analytical model. Each of these generic dimensions must allow a practically unlimited number of user-defined aggregation levels within any given consolidation path.

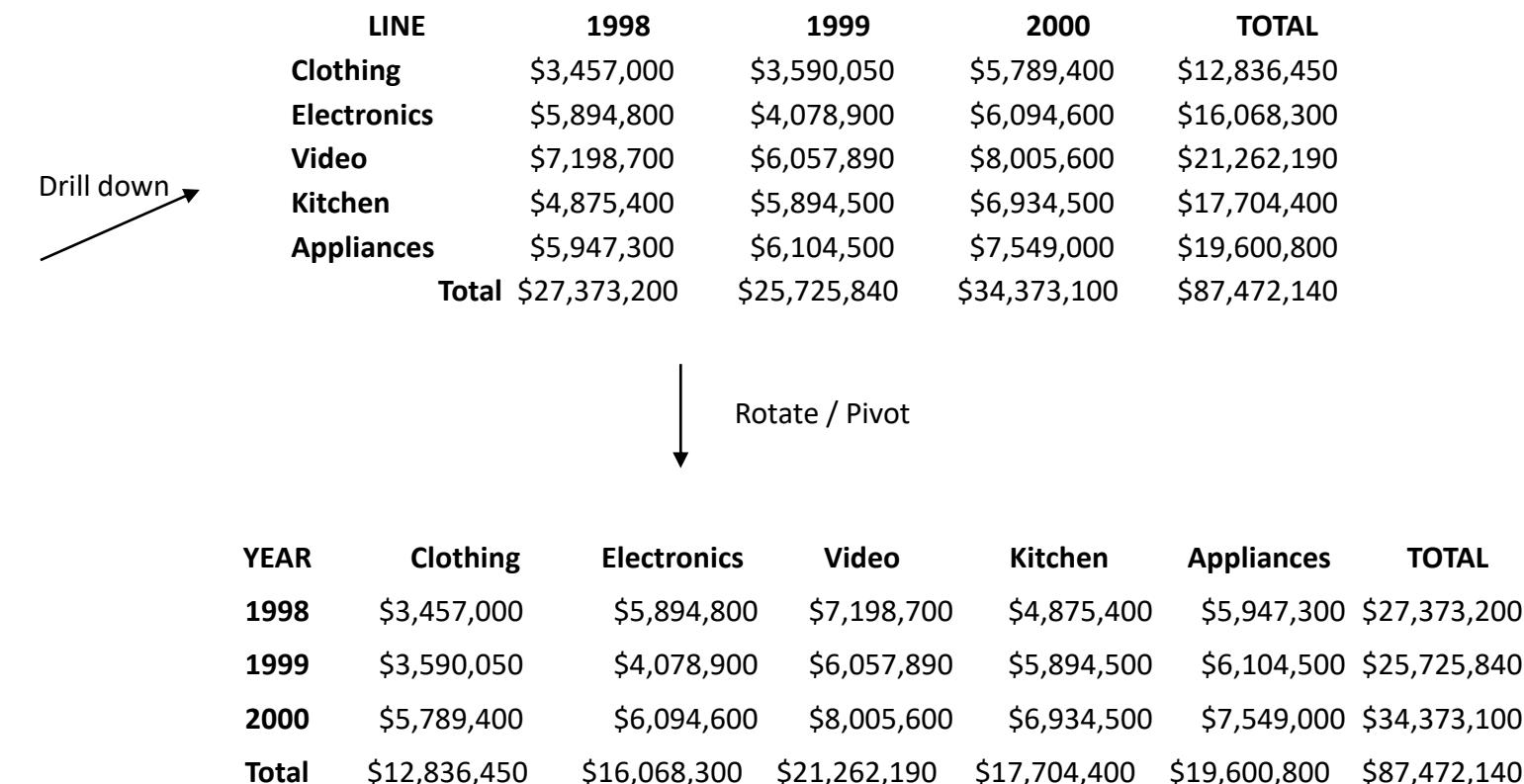
An Analysis Session



Typical Calculations

- Roll-ups to provide summaries and aggregations along the hierarchies of the dimensions.
- Drill-downs from the top level to the lowest along the hierarchies of the dimensions,
- in combinations among the dimensions.
- Simple calculations, such as computation of margins (sales minus costs).
- Share calculations to compute the percentage of parts to the whole.
- Algebraic equations involving key performance indicators.
- Moving averages and growth percentages.
- Trend analysis using statistical methods.

OLAP operations



The diagram illustrates three OLAP operations:

- Drill down:** An arrow points from the total sales row of the first table to the detailed product categories in the second table.
- Rotate / Pivot:** A large downward arrow points from the second table to the third table, indicating a transformation where columns become rows.
- Data Warehousing:** This is the base layer represented by the third table.

		LINE	1998	1999	2000	TOTAL		
LINE	TOTAL SALES	Clothing	\$3,457,000	\$3,590,050	\$5,789,400	\$12,836,450		
Clothing	\$12,836,450	Electronics	\$5,894,800	\$4,078,900	\$6,094,600	\$16,068,300		
Electronics	\$16,068,300	Video	\$7,198,700	\$6,057,890	\$8,005,600	\$21,262,190		
Video	\$21,262,190	Kitchen	\$4,875,400	\$5,894,500	\$6,934,500	\$17,704,400		
Kitchen	\$17,704,400	Appliances	\$5,947,300	\$6,104,500	\$7,549,000	\$19,600,800		
Appliances	\$19,600,800	Total	\$27,373,200	\$25,725,840	\$34,373,100	\$87,472,140		
Total	\$87,472,140	YEAR	Clothing	Electronics	Video	Kitchen	Appliances	TOTAL
		1998	\$3,457,000	\$5,894,800	\$7,198,700	\$4,875,400	\$5,947,300	\$27,373,200
		1999	\$3,590,050	\$4,078,900	\$6,057,890	\$5,894,500	\$6,104,500	\$25,725,840
		2000	\$5,789,400	\$6,094,600	\$8,005,600	\$6,934,500	\$7,549,000	\$34,373,100
		Total	\$12,836,450	\$16,068,300	\$21,262,190	\$17,704,400	\$19,600,800	\$87,472,140

Limitations of Other Tools

- Users need ability to analyse the data along multiple dimensions and their hierarchies rapidly
- Spreadsheets can be cumbersome to use, particularly for large volumes of data.
- Multidimensional data entered in spreadsheet has lot of redundancy
- It will require enormous effort to do create multidimensional view

Limitations of Other Tools

- SQL was originally meant to be end-user query language
- Except for very simple operations, the syntax is not easy to conceptualize for end-users
- The vocabulary is not suitable for analysis, comparisons are a challenge
- SQL is not good with complex calculations and time-series data.

Limitations of Other Tools

- A real-world analysis session requires many queries following one after the other.
- Each query may translate into a number of statements invoking full table scans, multiple joins, aggregations, groupings, and sorting.
- The overhead on the systems would be enormous and seriously impact the response times

Features of OLAP

Multidimensional analysis	Consistent performance	Fast response times for interactive queries
Drill-down and roll-up	Navigation in and out of details	Slice-and-dice or rotation
Multiple view modes	Easy scalability	Time intelligence (year-to-date, fiscal period)

Basic Features

Advanced Features

Powerful calculations	Cross-dimensional calculations	Pre-calculation or pre-consolidation
Drill-through across dimensions or details	Sophisticated presentation & displays	Collaborative decision making
Derived data values through formulas	Application of alert technology	Report generation with agent technology

CUBE Operator in SQL

- A cube aggregates the facts in each level of each dimension in a given OLAP schema
 - Data cubes are not "cubes" in the strictly mathematical sense because they do not have equal sides.
 - Most likely, there are more than 3 dimensions
- Major SQL vendors provide cube operator in their products
- Typical sequence for Cube computation:
 - Identify physical sources of data
 - Specify logical views built upon physical source
 - Build cube for specified measures and dimensions

Prescribed Text Books

	Author(s), Title, Edition, Publishing House
T1	Ponniah P, "Data Warehousing Fundamentals", Wiley Student Edition, 2012
T2	Kimball R, "The Data Warehouse Toolkit", 3e, John Wiley, 2013



BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Data Warehousing

M6: OLAP & Multidimensional Databases (MDDB)

T V Rao, BITS, Pilani (off-campus)



Multidimensional Databases

Why Multidimensional Database

- In 1960s, when a research scholar at MIT was doing analytical work on product sales, he realized that
 - he spent most of his time wrestling with reformatting the data for his analysis,
 - not on the statistical algorithms or the true data analysis
- Once he had modeled the data in a multidimensional form, he was able to report the data in many different formats
- By abstracting the data model from the data itself, the user could work with the data in an ad hoc fashion, asking questions that had not been formulated when developing the specifications

Multidimensional Database

- A multidimensional database, or MDB, stores dimensional information in a format called a *cube*.
- The basic concept of a cube is to precompute the various combinations of dimension values and fact values so they can be studied interactively
- Data in an MDB is accessed through an interface, which is often proprietary, although MDX (MultiDimensionalEXpressions) has gained wide acceptance as a standard
- MDBs support many statistical functions

A Motivating Example

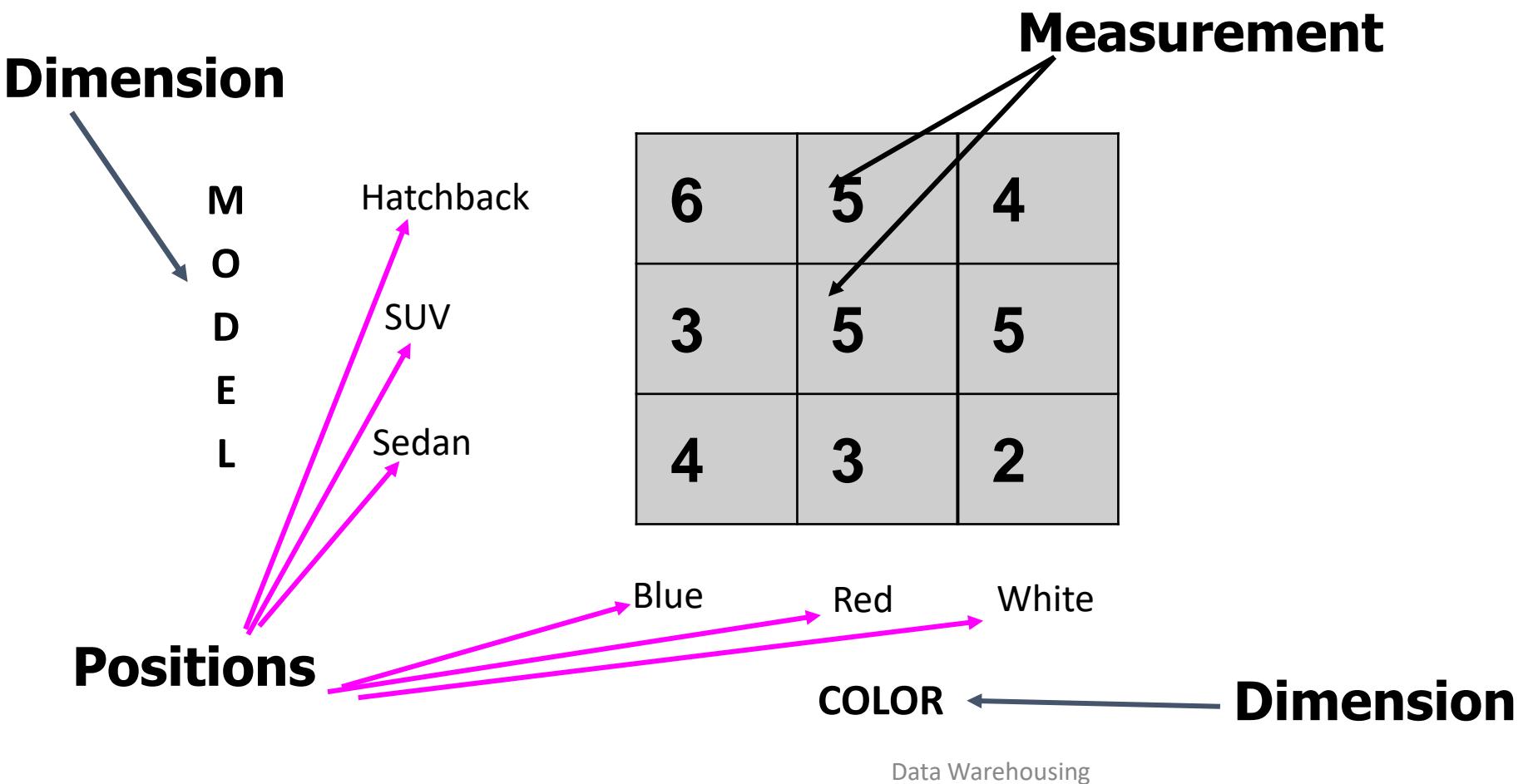
An automobile manufacturer wants to increase sale volumes by examining sales data collected throughout the organization. The evaluation would require viewing historical sales volume figures from multiple dimensions such as

- Sales volume by model
- Sales volume by color
- Sales volume by dealer
- Sales volume over time

Sales Data in Relational Form

MODEL	COLOR	SALES VOLUME
Hatchback	BLUE	6
Hatchback	RED	5
Hatchback	WHITE	4
SUV	BLUE	3
SUV	RED	5
SUV	WHITE	5
SEDAN	BLUE	4
SEDAN	RED	3
SEDAN	WHITE	2

Multidimensional Structure



Comments on Multidimensional Structure

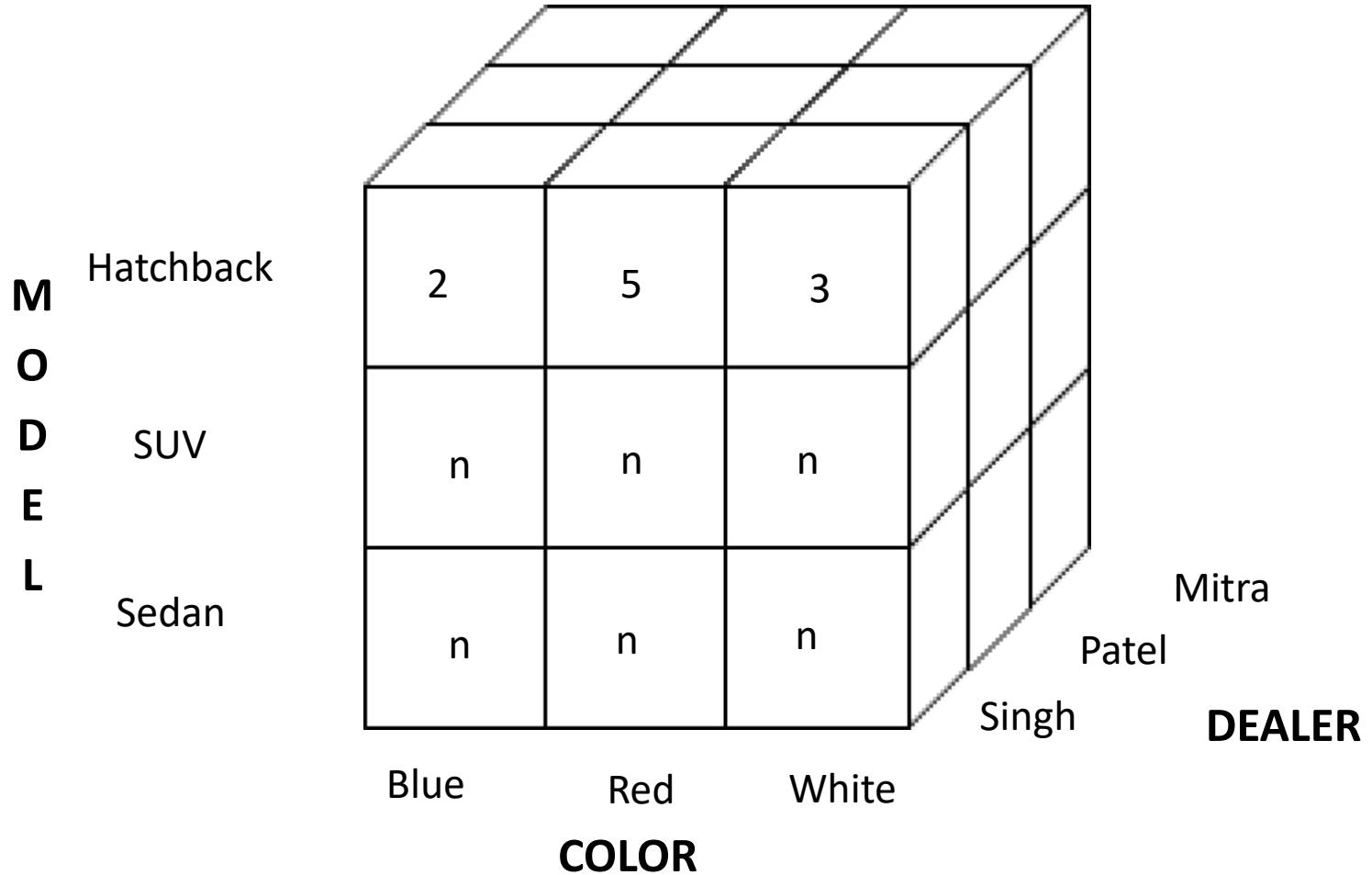


- The relational structure tells us nothing about the possible contents of those fields.
- The structure of the array, on the other hand, tells us not only that there are two dimensions, COLOR and MODEL, but it also presents all possible values of each dimension as positions along the dimension.
- Because of this structured presentation, all possible combinations of perspectives containing a specific attribute (the color BLUE, for example) line up along the dimension position for that attribute.
- This makes data browsing and manipulation highly intuitive to the end-user. As a result, this "intelligent" array structure lends itself very well to data analysis.

Let us add Dealer to the table

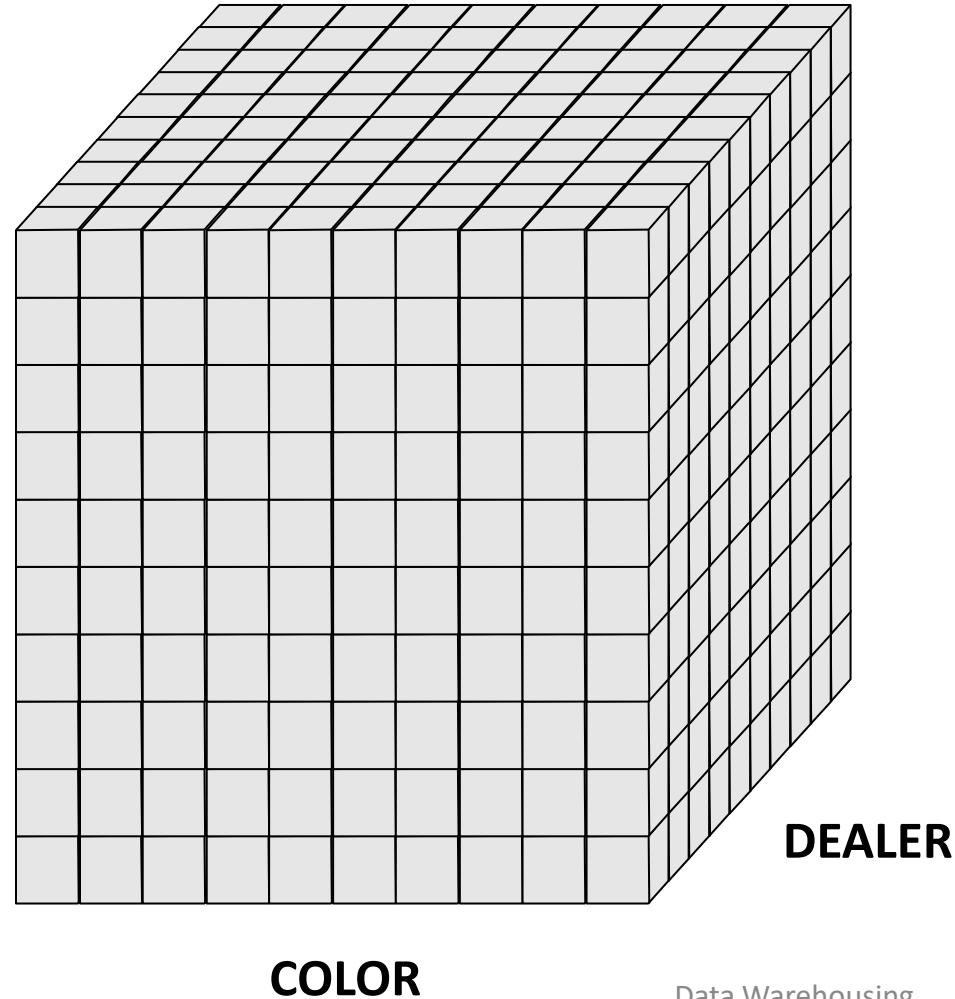
MODEL	COLOR	DEALERSHIP	VOLUME
Hatchback	BLUE	Mitra	6
Hatchback	BLUE	Patel	6
Hatchback	BLUE	Singh	2
Hatchback	RED	Mitra	3
Hatchback	RED	Patel	5
Hatchback	RED	Singh	5
Hatchback	WHITE	Mitra	2
Hatchback	WHITE	Patel	4
Hatchback	WHITE	Singh	3
SUV	BLUE	Mitra	2
SUV	BLUE	Patel	3
SUV	BLUE	Singh	2
SUV	RED	Mitra	7
SUV	RED	Patel	5
SUV	RED	Singh	2
SUV	WHITE	Mitra	4
SUV	WHITE	Patel	5
SUV	WHITE	Singh	1
SEDAN	BLUE	Mitra	6
SEDAN	BLUE	Patel	4
SEDAN	BLUE	Singh	2
SEDAN	RED	Mitra	1
SEDAN	RED	Patel	3
SEDAN	RED	Singh	4
SEDAN	WHITE	Mitra	2
SEDAN	WHITE	Patel	2
SEDAN	WHITE	Singh	3

Multidimensional Structure



Multidimensional Structure

M
O
D
E
L



If each dimension has 10 positions the relational table requires $10*10*10$ i.e. 1000 records

Performance Advantages with MDB

Volume figure when car type = SEDAN, color=BLUE, & dealer=PATEL?

- RDBMS – all 1000 records might need to be searched to find the right record
- MDB has more ‘knowledge’ about where the data lies
 - Maximum of 30 position searches
- Average case
 - MDB 15 vs. 500 for RDBMS

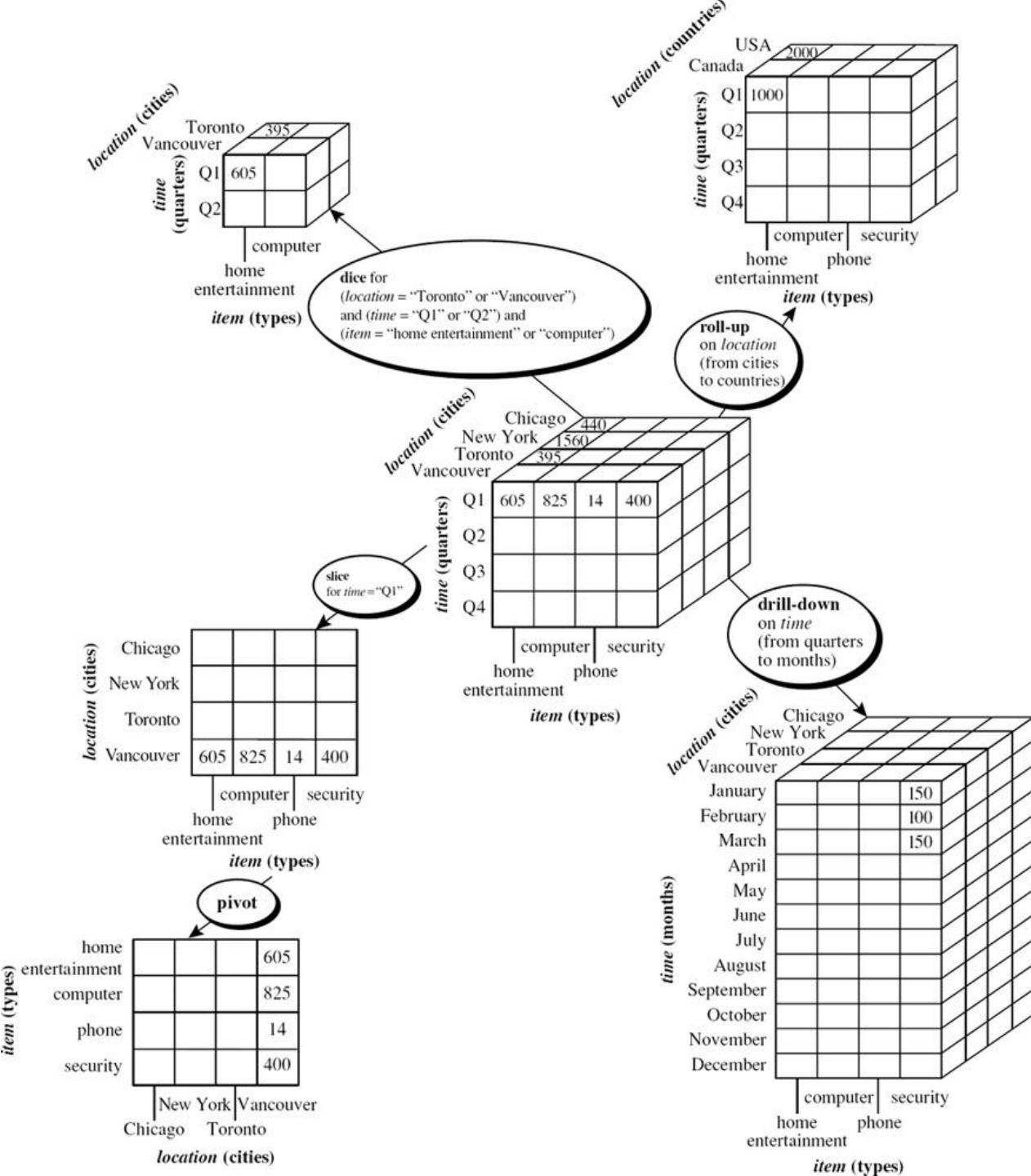
Performance Advantages with MDB

- To generalize the performance advantage
 - In case of RDBMS, size of search space gets multiplied, each time a new dimension is added; accordingly access time is affected
 - In case of MDB, the search space increases by the size of new dimension, each time a new dimension is added.
- At what cost?
 - MDB is a separate proprietary implementation from SQL
 - Since all business data is in RDBMS, the MDB has to be precomputed. Larger the data, more the dimensions, higher the precomputation effort.
 - Higher the interval of precomputation, higher the latency in MDB

OLAP Operations

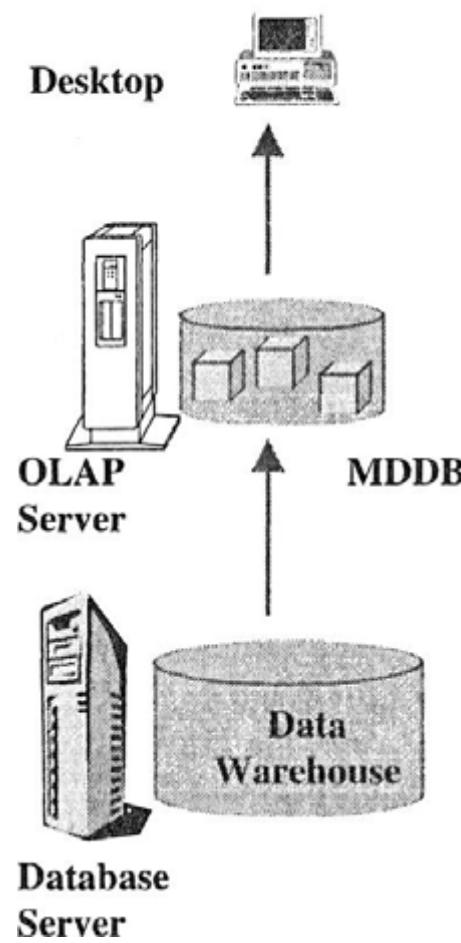
- Roll up (drill-up): summarize data
 - *by climbing up hierarchy or by dimension reduction*
- Drill down (roll down): reverse of roll-up
 - *from higher level summary to lower level summary or detailed data, or introducing new dimensions*
- Slice and dice: *project and select*
- Pivot (rotate):
 - *reorient the cube, visualization, 3D to series of 2D planes*
- Other operations
 - *drill across: involving (across) more than one fact table*
 - *drill through: through the bottom level of the cube to its back-end relational tables (using SQL)*

OLAP Operations

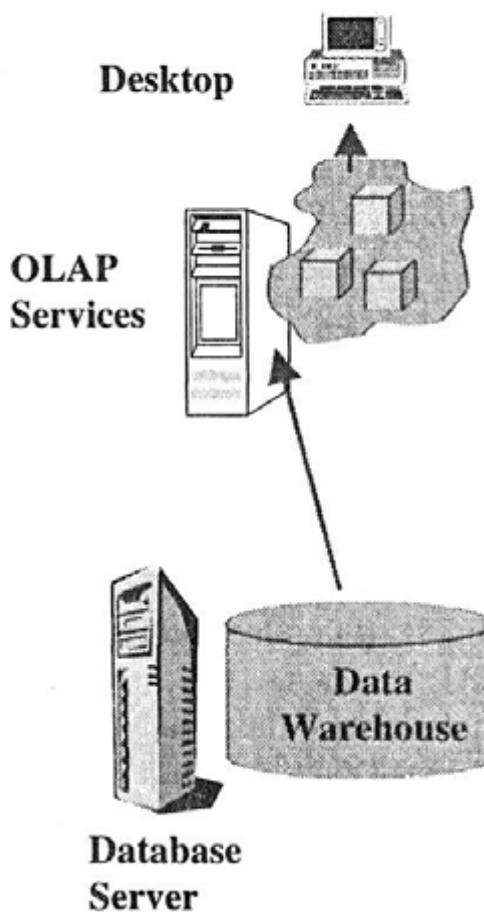


Distinct OLAP models

MOLAP



ROLAP



Data Warehousing

Data Storage

ROLAP	MOLAP
Data stored as relational tables in the warehouse.	Data stored as relational tables in the warehouse.
Detailed and light summary data available.	Various summary data kept in proprietary databases (MDBs)
Very large data volumes.	Moderate data volumes.
All data access from the warehouse storage.	Summary data access from MDB, detailed data access from warehouse.

Underlying Technologies

ROLAP	MOLAP
Use of complex SQL to fetch data from warehouse.	Creation of pre-fabricated data cubes by MOLAP engine.
ROLAP engine in analytical server creates data cubes on the fly.	Proprietary technology to store multidimensional views in arrays, not tables.
Multidimensional views by presentation layer.	High speed matrix data retrieval. Sparse matrix technology to manage data sparsity in summaries.

Functions and Features

ROLAP	MOLAP
Known environment and availability of many tools.	Faster access.
Limitations on complex analysis functions.	Large library of functions for complex calculations.
Drill-through to lowest level easier. Drill-across not always easy.	Easy analysis irrespective of the number of dimensions. Extensive drill-down and slice-and-dice capabilities.

HOLAP (Hybrid OLAP)

The intermediate architecture type, *HOLAP*, aims at mixing the advantages of both basic solutions. It takes advantage of the standardization level and the ability to manage large amounts of data from ROLAP implementations, and the query speed typical of MOLAP systems. HOLAP has the largest amount of data stored in an RDBMS, and a multidimensional system stores only the information users most frequently need to access. If that information is not enough to solve queries, the system will transparently access the part of the data managed by the relational system. Important market actors have adopted HOLAP solutions to improve their platform performance.

Prescribed Text Books

Author(s), Title, Edition, Publishing House	
T1	Ponniah P, "Data Warehousing Fundamentals", Wiley Student Edition, 2012
T2	Kimball R, "The Data Warehouse Toolkit", 3e, John Wiley, 2013

References

Author(s), Title, Edition, Publishing House	
	Star Schema: The Complete Reference by Christopher Adamson McGraw-Hill/Osborne
	Kenan Technologies, An introduction to Multidimensional Database Technology
	Data Mining: Concepts and Techniques, Third Edition by Jiawei Han, Micheline Kamber and Jian Pei Morgan Kaufmann Publishers

Thank You