Big Data Technologies

Advanced Analytics

Lionel Fillatre
Polytech Nice Sophia
lionel.fillatre@univ-cotedazur.fr

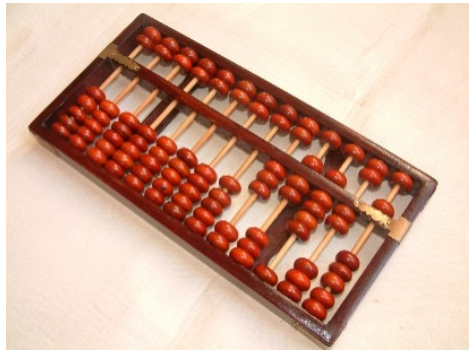
Outlines

- Historical Perspective
- Data Lake
- Analytics
- Advanced Analytics
- Mllib
- MLlib Guide
- Conclusion

Historical Perspective

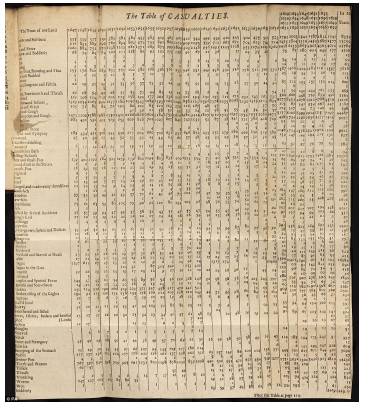
History of Data Processing

- 18,000 BCE: The earliest examples we have of humans storing and analyzing data are the tally sticks. Palaeolithic people would compare sticks and notches to carry out rudimentary calculations, enabling them to make predictions such as how long their food supplies would last.



- 2400 BC: the abacus – the first dedicated device constructed specifically for performing calculations – comes into use in Babylon

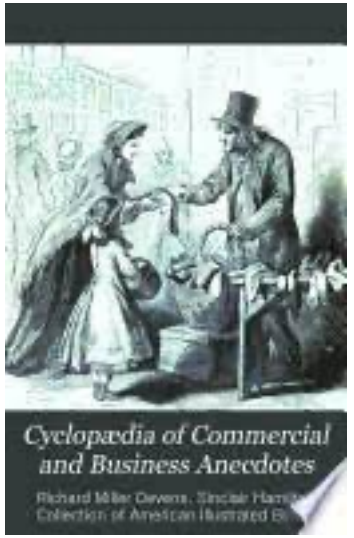
- 1663: In London, John Graunt carries out the first recorded experiment in statistical data analysis (bubonic plague).

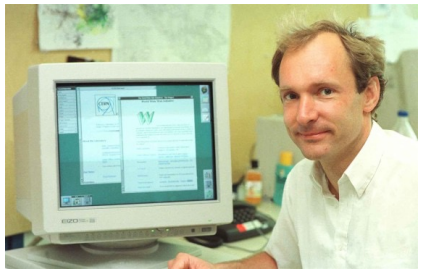


https://www.weforum.org/agenda/2015/02/a-brief-history-of-big-data-everyone-should-read/

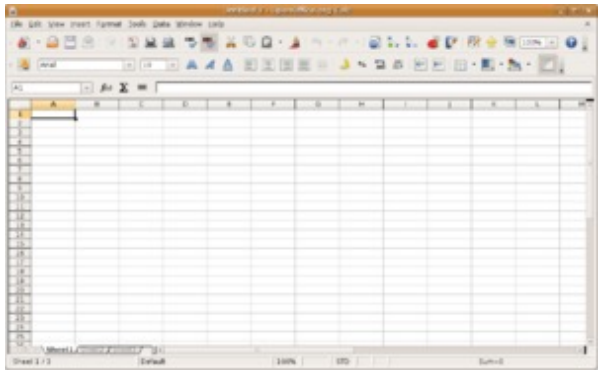
History of Data Processing

- 1865: The term « business intelligence » is used by Richard Millar Devens, describing how the banker Henry Furnese achieved an advantage over competitors by collecting and analyzing information.
- 1880: In 1881 a young engineer employed by the US Census Bureau – Herman Hollerith – produces what will become known as the Hollerith Tabulating Machine. The company he founds will go on to become known as IBM.
- 1928: Fritz Pfleumer, a German-Austrian engineer, invents a method of storing information magnetically on tape.
- 1990s: The internet ! Sir Tim Berners Lee created hyperlinks and hypertext, enabling data sharing worldwide.





The Drive for Self-Service Data

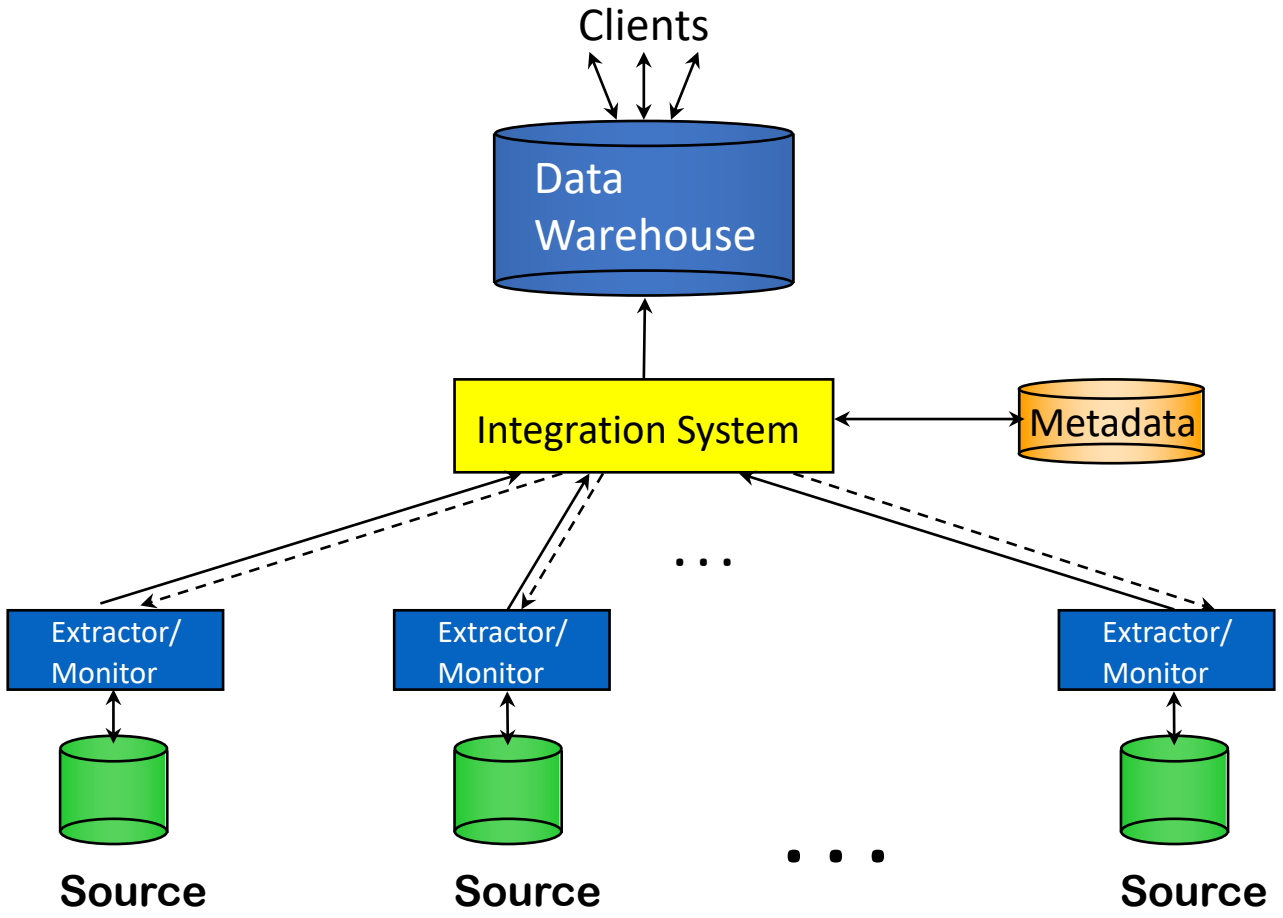


- The first step in the self-service data revolution was the spreadsheet. It allowed non-developers to work with data directly.
  - They did not scale beyond small amounts of data and could address only a small subset of the problems that analysts wanted to address.

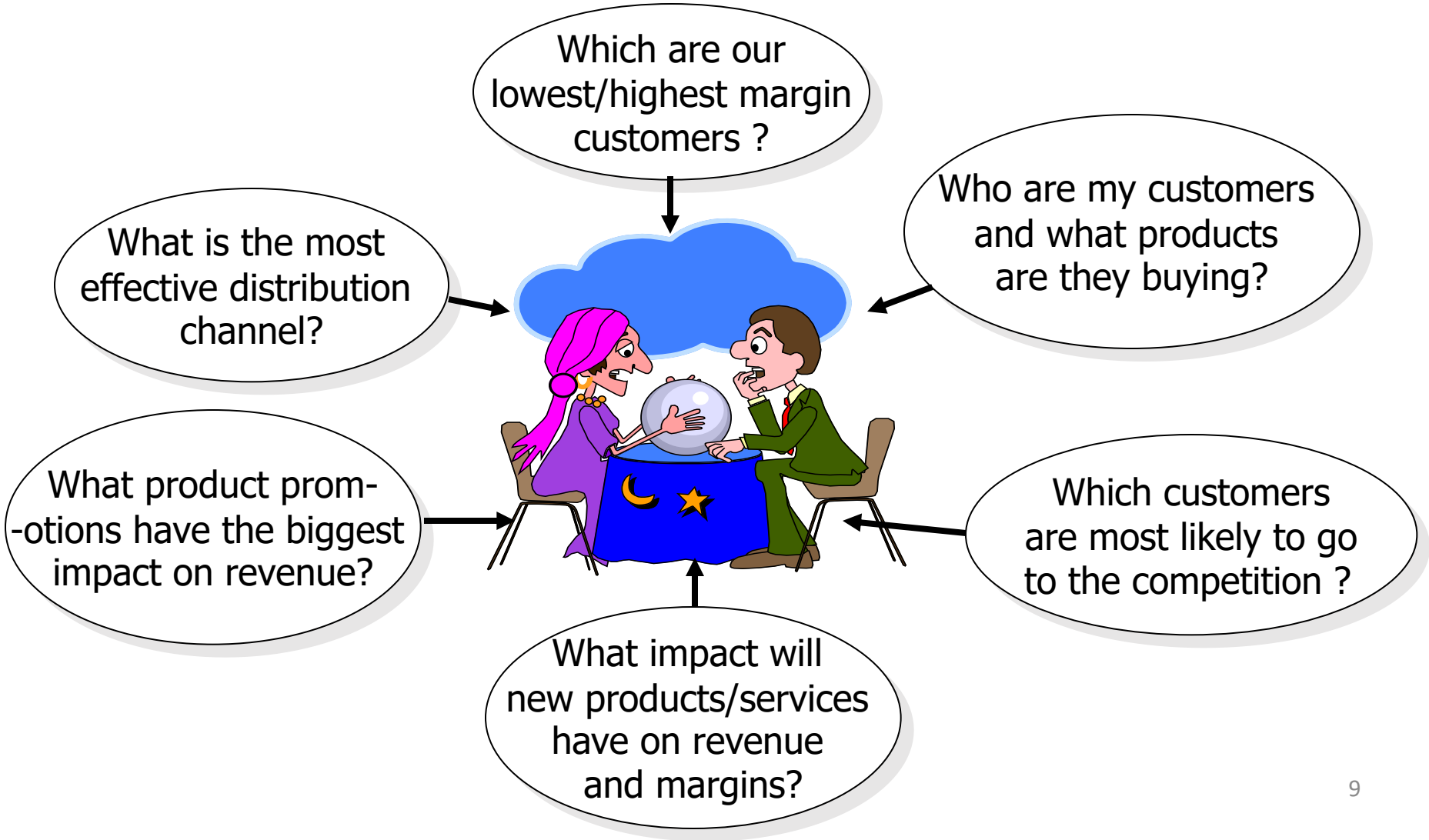
- Meanwhile, companies began to realize that the data, not the applications, was the crown jewel.
  - Data had to be carefully managed, checked for consistency, and backed up.
  - Instead of each program having to develop these capabilities itself, they were extracted and provided by a new class of systems called database management systems (DBMS).
  - These systems did not contain programming logic and existed purely to manage the data.
  - A somewhat standard language called Structured Query Language (SQL) emerged.
  - Using the SQL language, the users could write their own queries and do their own analysis of data.

The Birth of Data Warehousing

- The original vision was to create a « warehouse » that would store all the data and thus all the history about an enterprise, and make it available for analytics.



Why Data Warehousing?



So What Is a Data Warehouse?

- **Definition:** A single, complete and consistent store of data obtained from a variety of different sources made available to end users in a what they can understand and use in a business context.

- **Online Analytical Processing (OLAP)** is a category of software that allows users to analyze information from multiple database systems at the same time

- Main features:
  - Aggregation -- (total sales, percent-to-total)
  - Comparison -- Budget vs. Expenses
  - Ranking -- Top 10, quartile analysis
  - Visualization

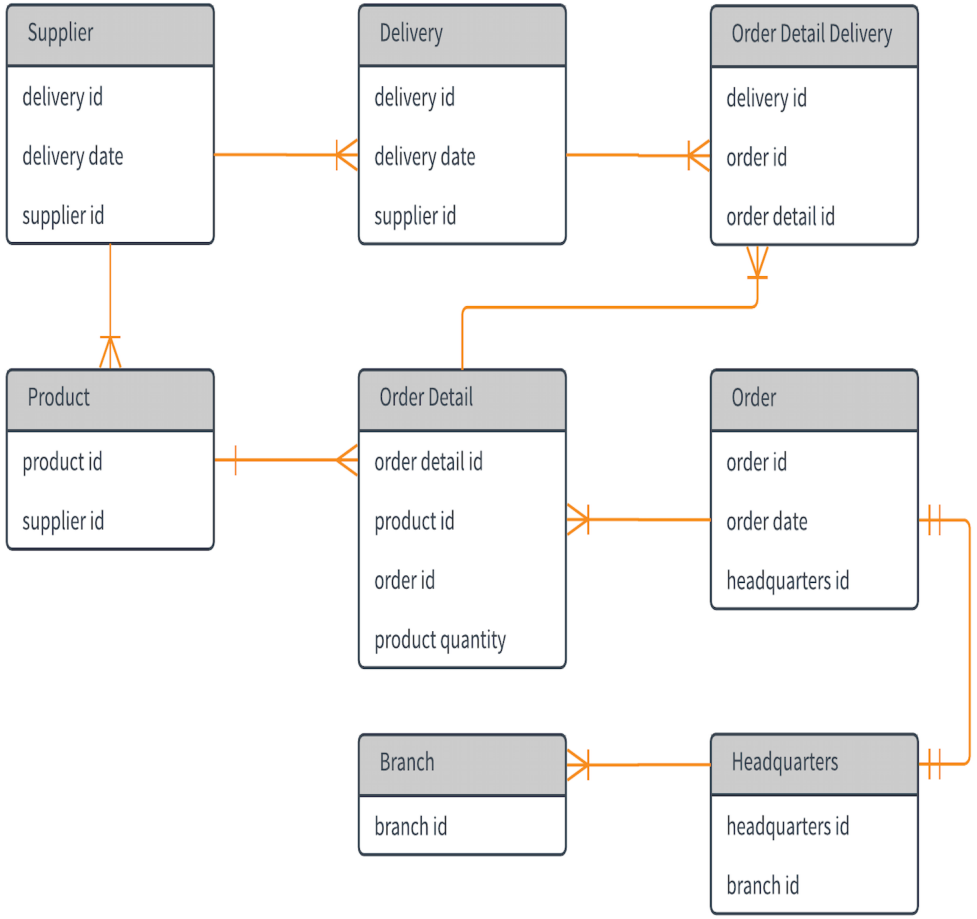
Comparison with OLTP

- By comparison, an **OLTP (On-Line Transaction Processor)** or operational system is used to deal with the everyday running of one aspect of an enterprise:
  - Online banking
  - Online airline ticket booking
  - Etc.

- OLTP systems are usually designed independently of each other and it is difficult for them to share information.

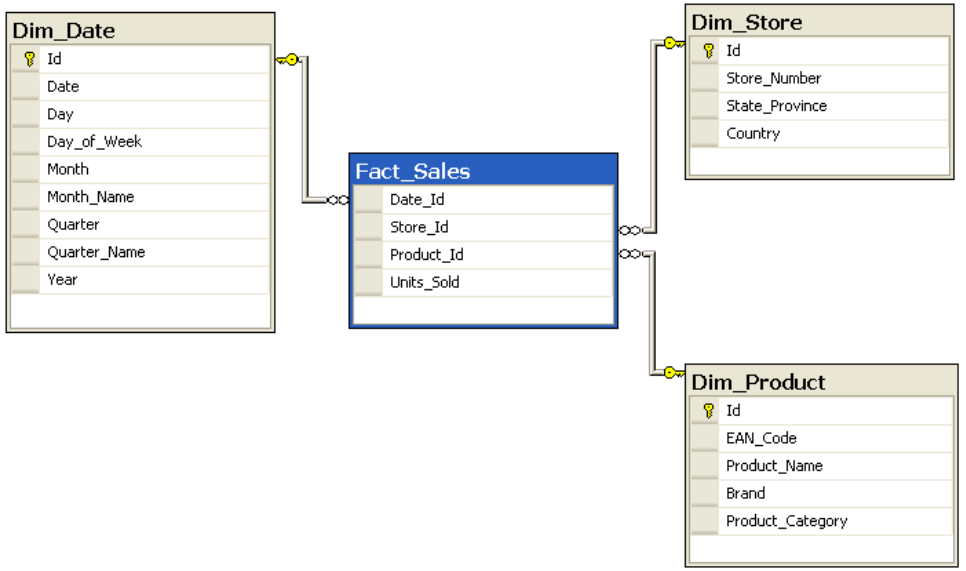
Design Differences

Operational System (OLTP)



Entity Relationship Diagram

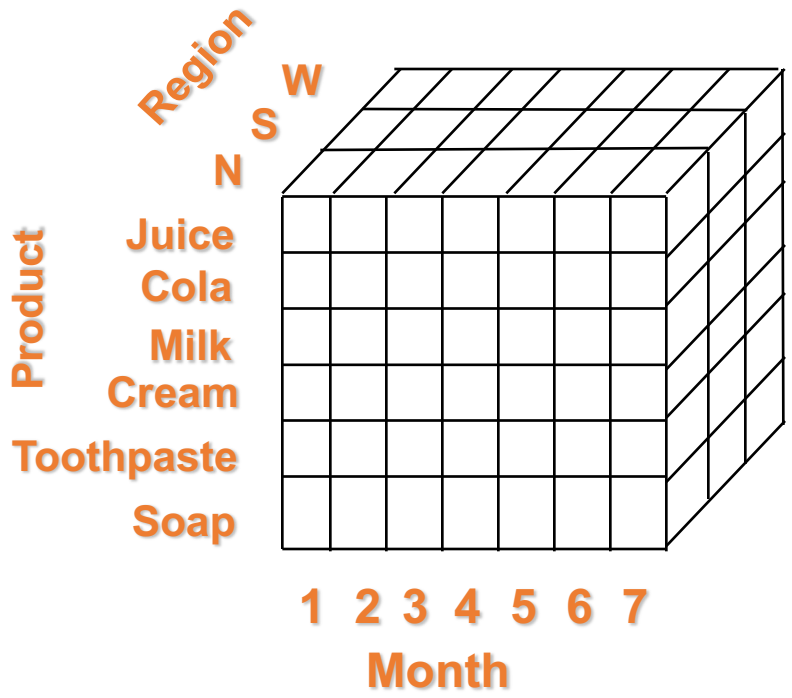
Data Warehouse (OLAP)



Star Schema

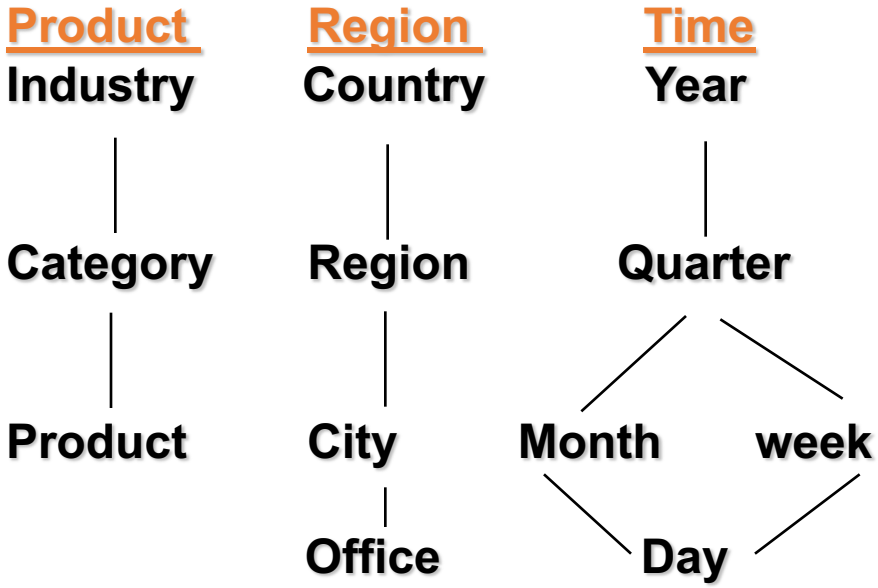
Multi-dimensional Data

- Measure - sales (actual, plan, variance)

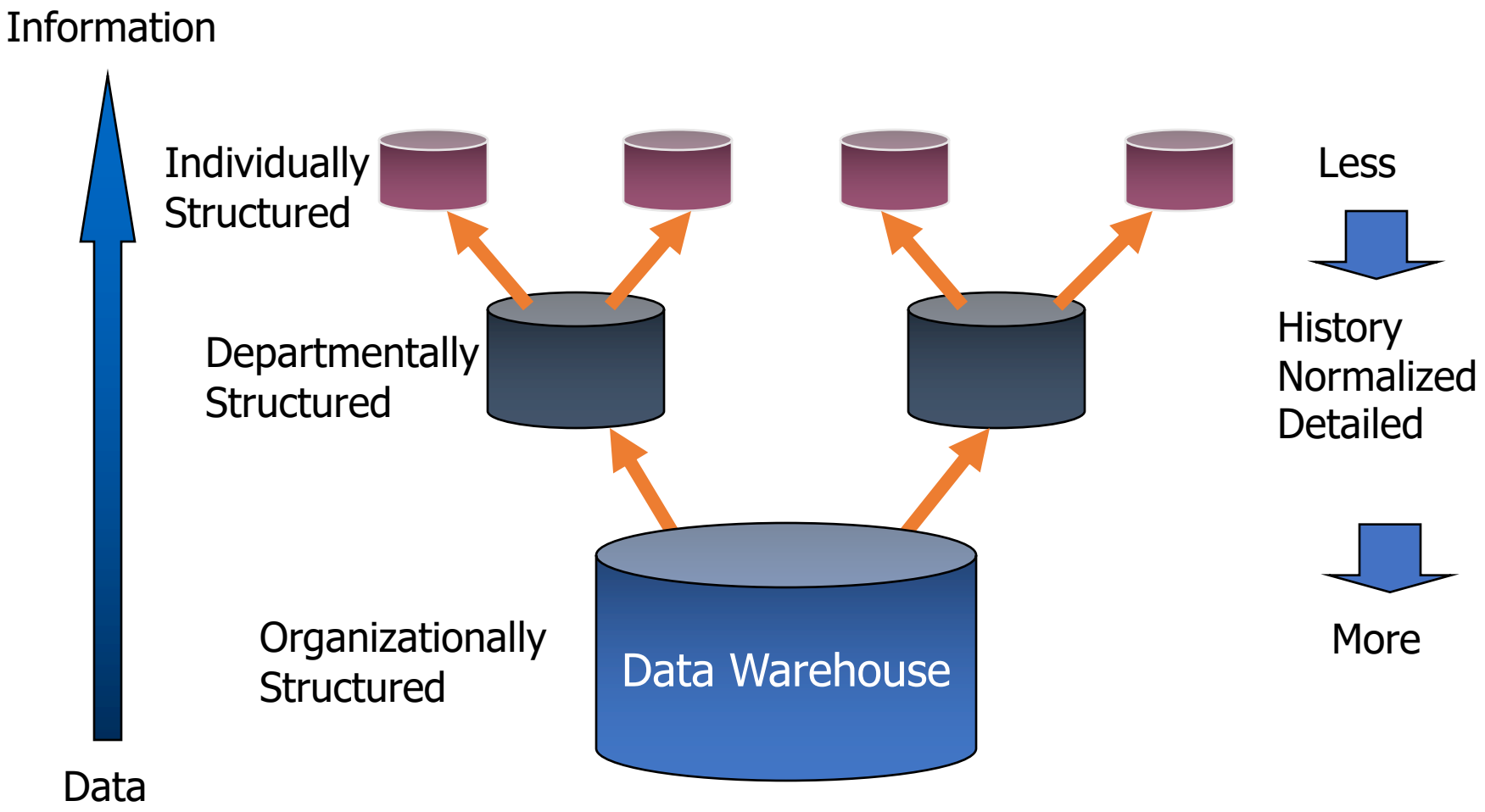


Dimensions: Product, Region, Time

Hierarchical summarization paths



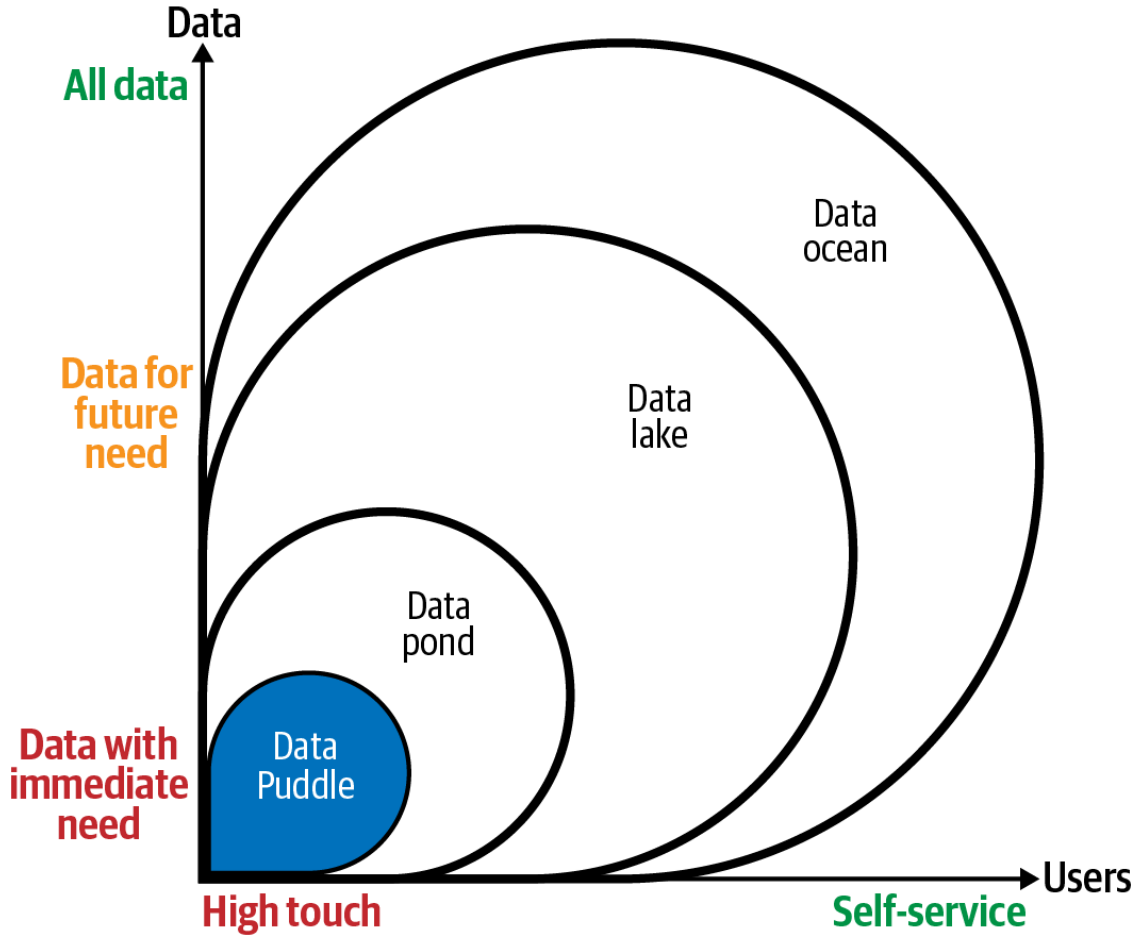
From the Data Warehouse to Data Marts



Data Lake

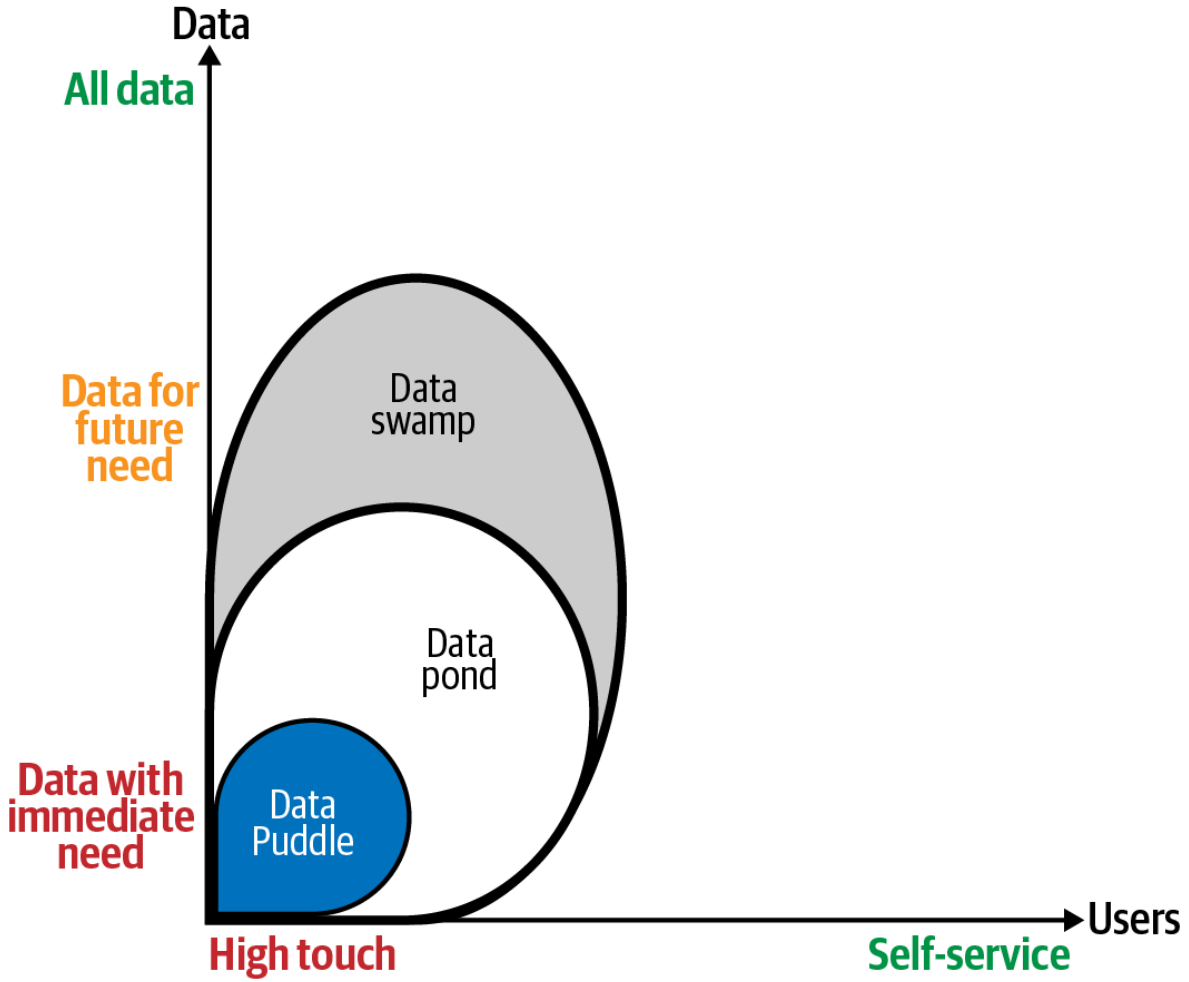
What is a Data Lake?

- **Data puddle**: A single-purpose or single-project data mart built using big data technology
- **Data pond**: A collection of data puddles or an offload of an existing data warehouse
- **Data lake**: A data lake supports self-service and it contains data that business users might possibly want, even if no project requires it at this time.
- **Data ocean**: This expands self-service data and data-driven decision-making to all enterprise data, wherever it may be, regardless of whether it was loaded into the data lake.



Danger: The Data Swamp

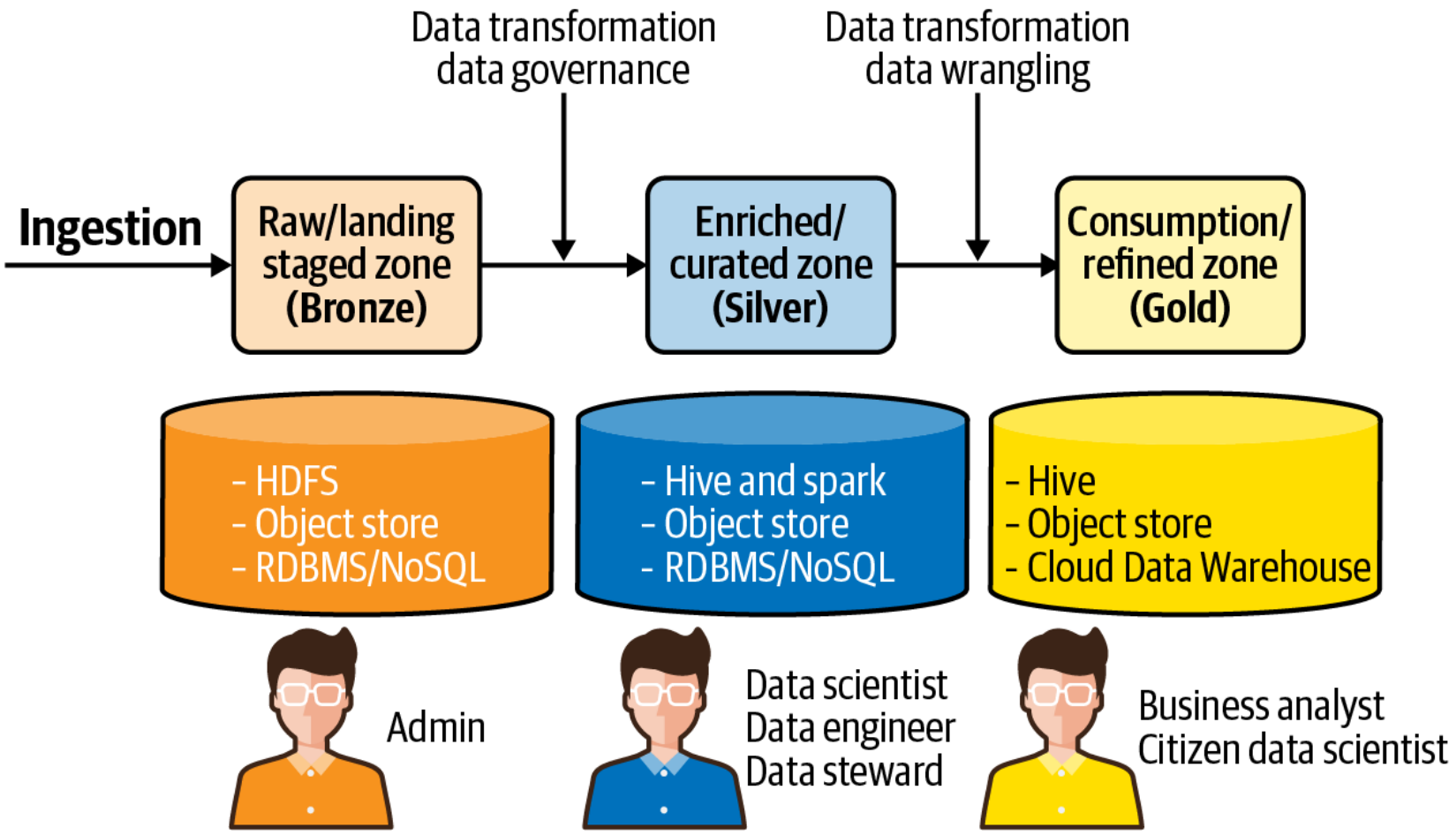
- While data lakes always start out with good intentions, sometimes they take a wrong turn and end up as data swamps.
- A **data swamp** is a data pond that has grown to the size of a data lake but failed to attract a wide data analyst community, usually because of a lack of self-service and governance facilities.
- At best, the data swamp is used like a data pond, and at worst, it is not used at all.



Types of users in a cloud data lake

User type	Purpose
Data steward	Oversees and manages data quality, governance, and compliance.
Business analyst	Uses a BI tool to access reports and dashboards.
Data analyst	Develops ad hoc queries using SQL on data in staging or enriched zone.
(Citizen) data engineer	Implements business logic using an integration approach described previously. Citizen data engineers are using more UI-based drag-and-drop options in lieu of coding.
(Citizen) data scientist	Trains and validates ML models using IDEs and notebooks such as Jupyter, Apache Zeppelin, or RStudio. Citizen data scientists are not formally trained in ML but may use SaaS ML products.
MLOps engineer	Versions and deploys the appropriate ML model into production. This may require working with the DevOps team.

Data Analysis Pipeline in a Data Lake

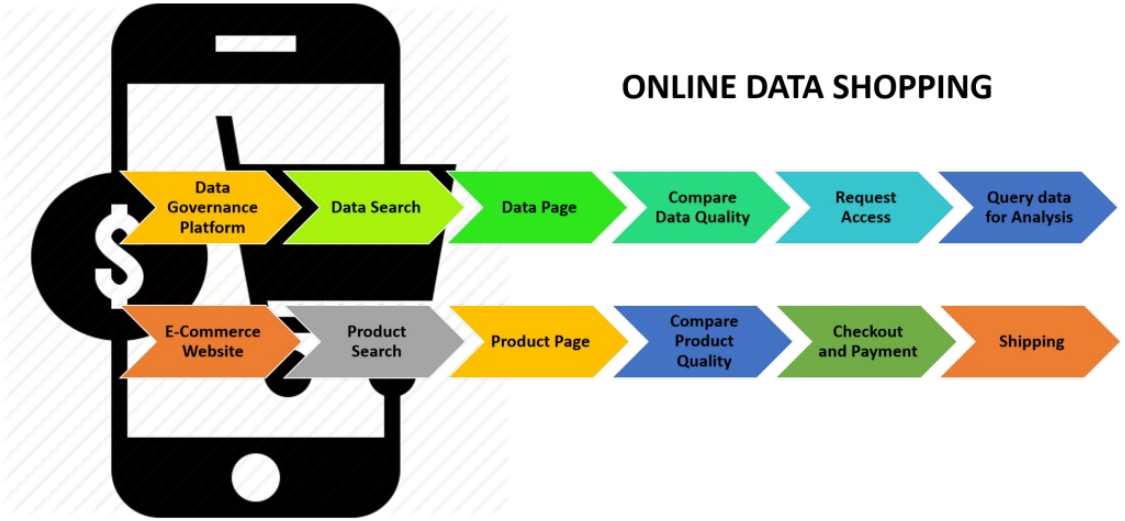


Bronze, silver and gold form the medallion architecture (common terminology in Data Lakes)

On Line Data Shopping

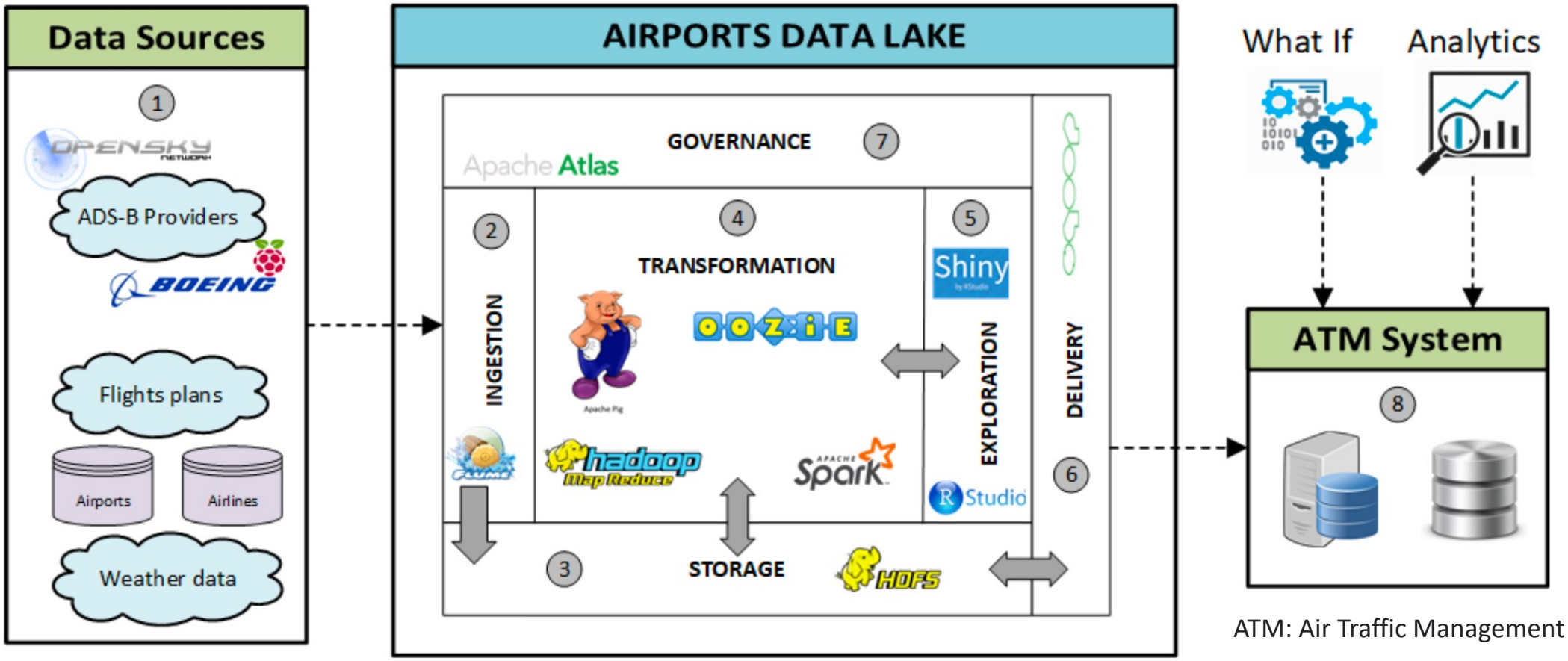
- For the success of any enterprise data lake, it is essential that data citizens should be able to do the following tasks with ease
  - Search, understand, and use that data to derive useful information from it.
  - Should be able to compare the trustworthiness of data.
  - Should be easily able to know how to get access to any data and from whom.

- In simple words, getting the data from the data lake should be like a shopping experience.



Data processing is compared to online shopping

Example: Airports Datalake

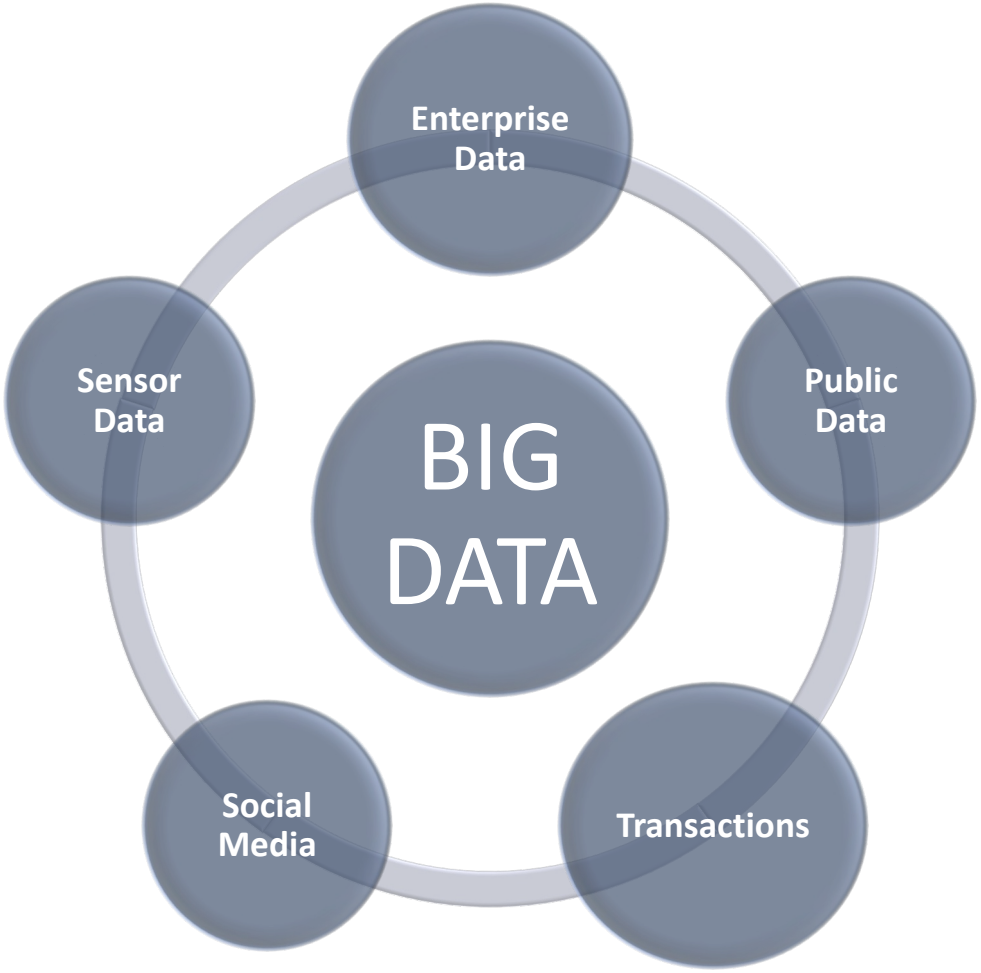


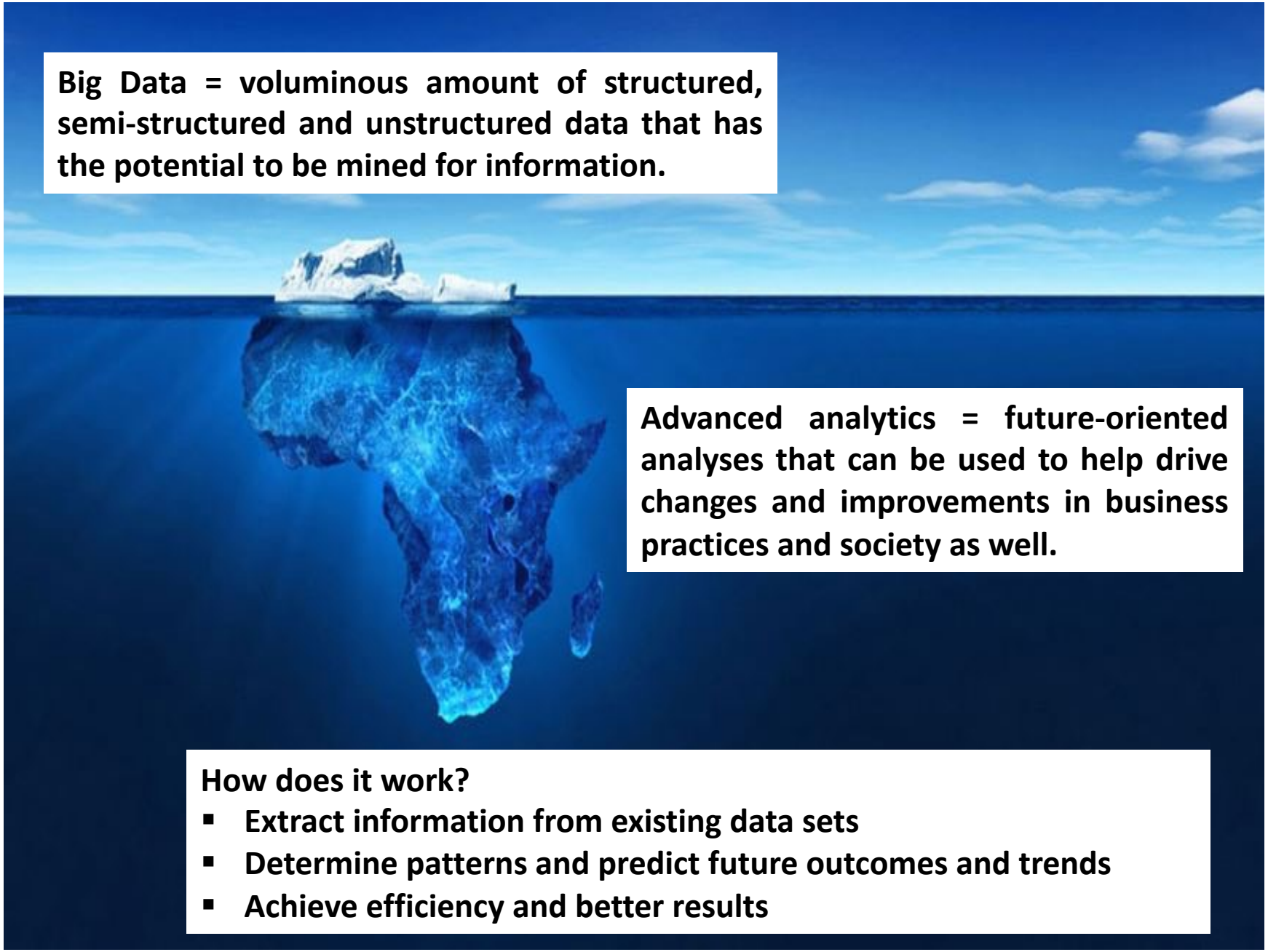
ATM: Air Traffic Management

Integrating flight-related information into a (Big) data lake, IEEE/AIAA 36th Digital Avionics Systems Conference (DASC), 2017

Analytics

Main Sources of Data




Big Data = voluminous amount of structured, semi-structured and unstructured data that has the potential to be mined for information.
Advanced analytics = future-oriented analyses that can be used to help drive changes and improvements in business practices and society as well.
How does it work?
Extract information from existing data sets
Determine patterns and predict future outcomes and trends
Achieve efficiency and better results

Analytics

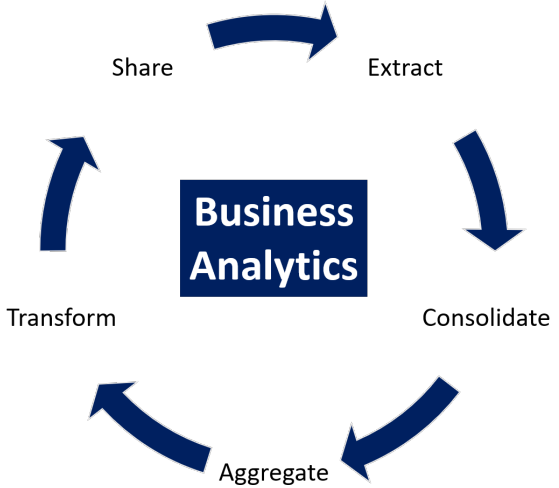
- Analytics refers to the skills, technologies, applications and practices for continuous iterative exploration and investigation of data to gain insight and drive business planning.

- Analytics consists of two major areas: **Business Intelligence** and **Advanced Analytics**.

Business Intelligence

- Business Intelligence traditionally focuses on using a consistent set of metrics to **measure past performance and guide business planning**.

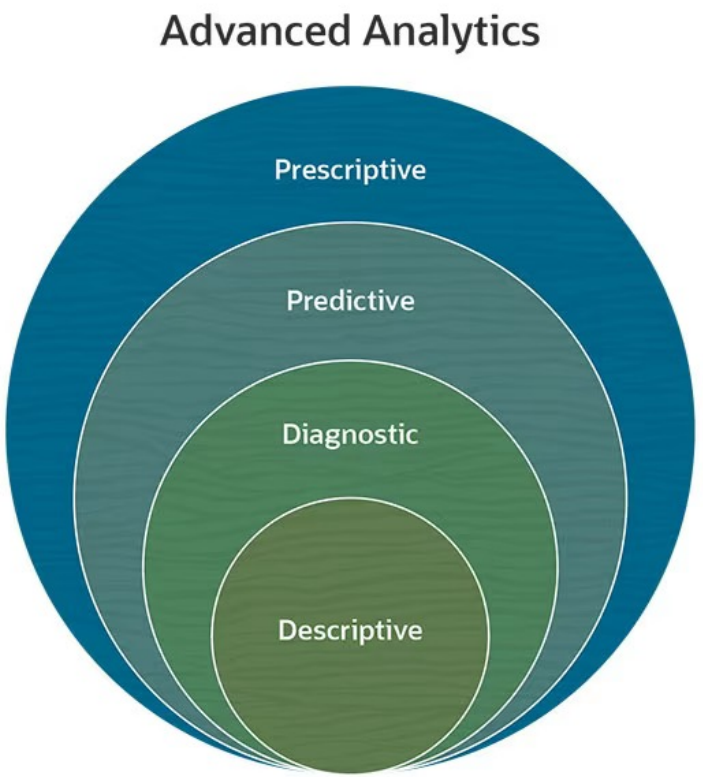
- Business Intelligence consists of querying, reporting, OLAP (online analytical processing), and can answer questions including "what happened", "how many", and "how often."



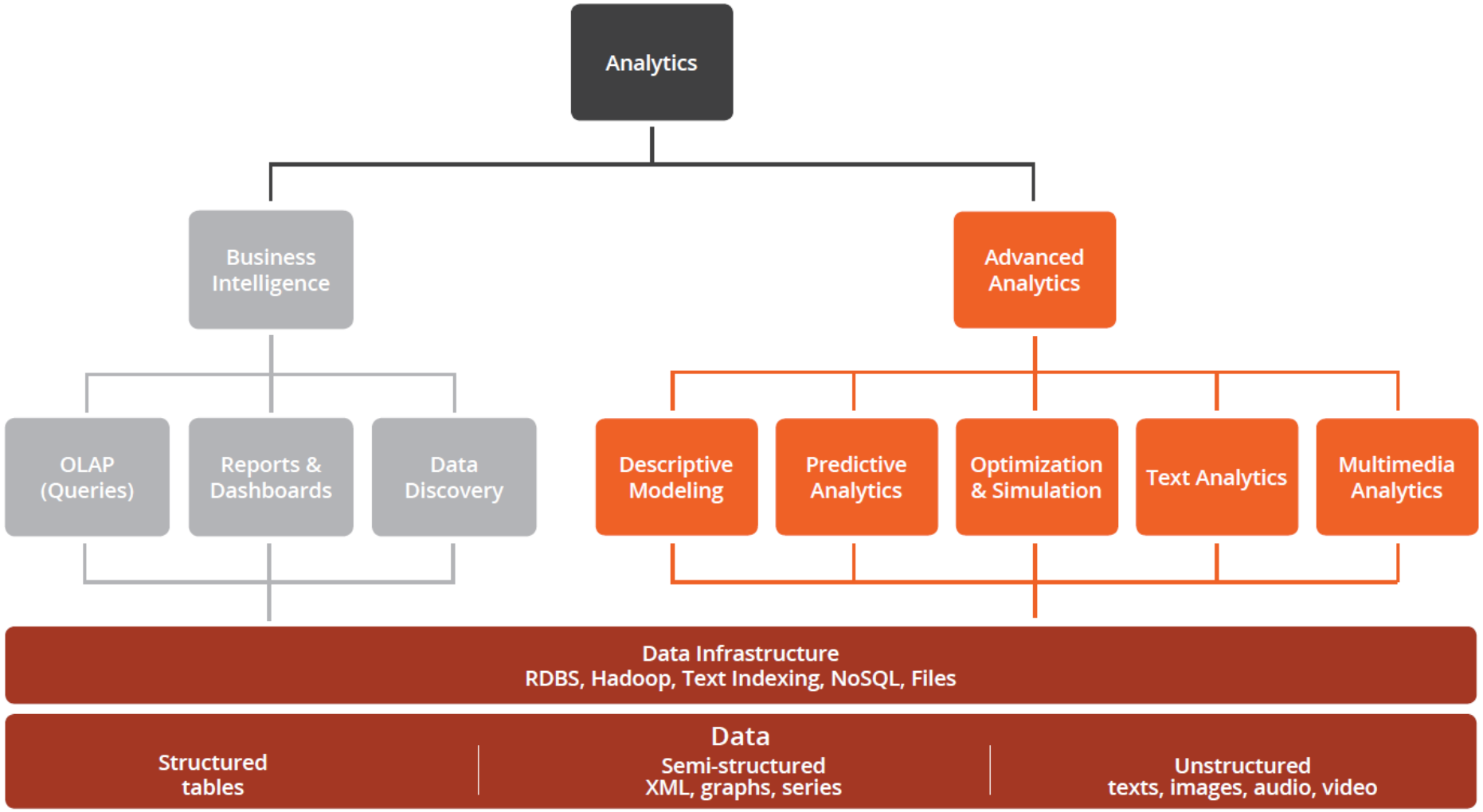
Advanced Analytics

- Advanced Analytics goes beyond Business Intelligence by using sophisticated modeling techniques to **predict future events or discover patterns** which cannot be detected otherwise.

- Advanced analytics can answer question including "why is this happening", "what if these trends continue", "what will happen next" (prediction), "what is the best that can happen" (prescription)

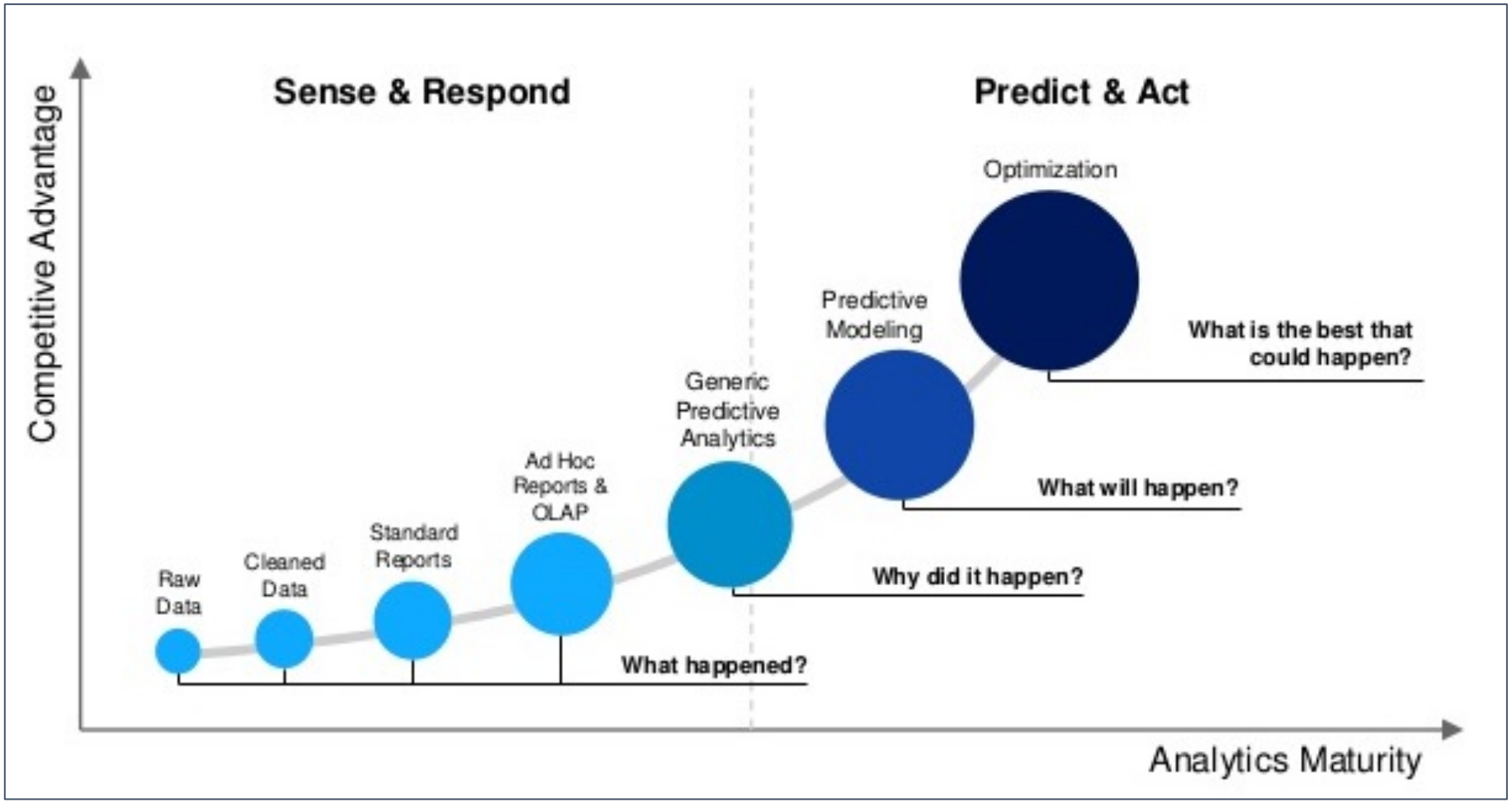


Analytics and Some Subfields



Advanced Analytics

Advanced Analytics



How Predictive Analytics is Used in Industry

Healthcare

Predict likelihood of disease to begin early treatment; identify clinical trial outcomes.

Insurance

Identify unusual transactions for fraud prevention. Rate / Score the risk that is to be insured.

Banking

Identify key behaviors of customers likely to leave the bank; improve credit risk analysis.

Utilities

Forecast demand and usage for seasonal operations; provide anticipated resources.

CRM Marketing

Identify potential leads among existing customers and intelligently market to them based on individual preferences and histories

Government

Predict community movement and trends that affect taxing districts; anticipate revenue.

Retail

Product suggestions based on past purchases; inventory planning; selection of store locations based on demographics.

Telco

Forecast demand on system load for capacity planning and customer scale. Reduce customer churn. Keep influencer customers.

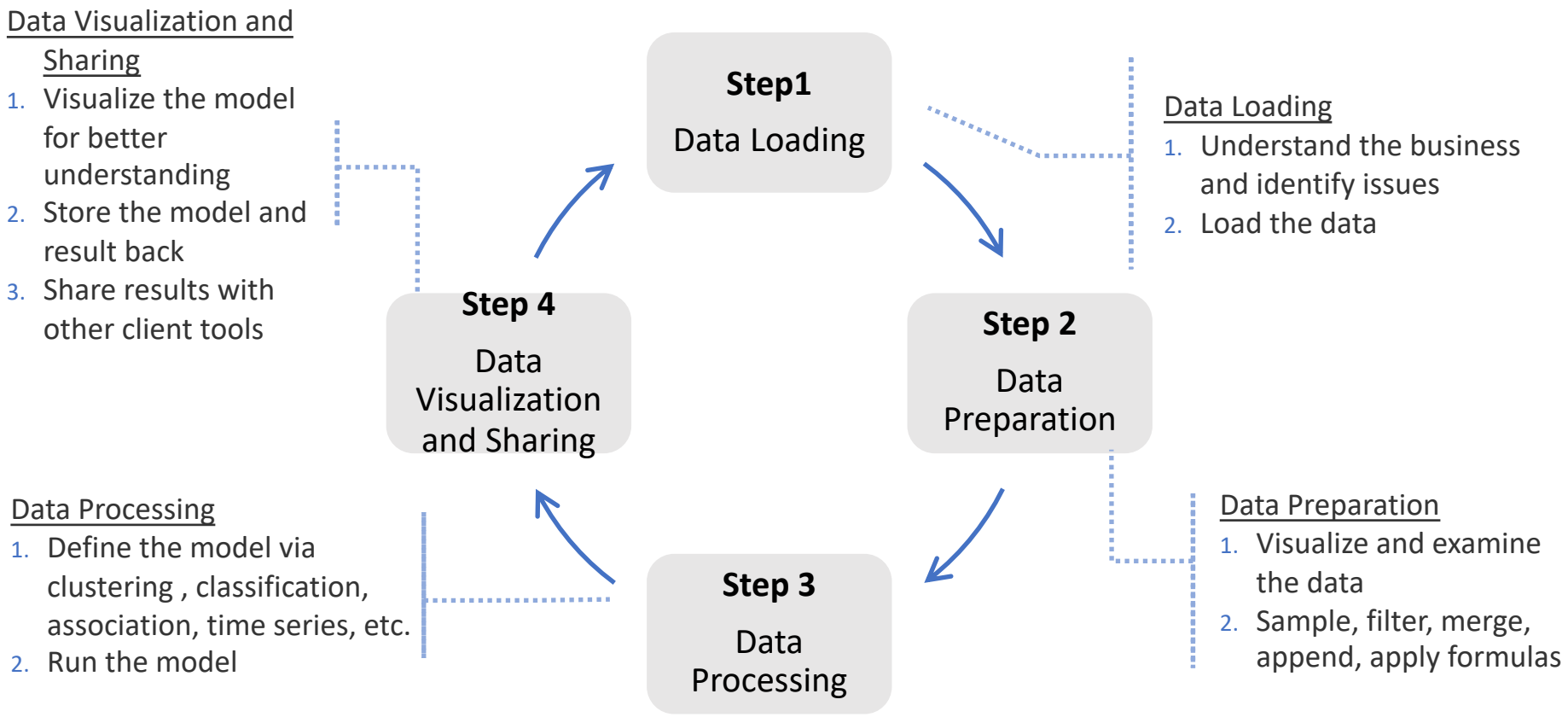
Common Use Cases in Advanced Analytics

- Recommend friends/dates/products
- Classify content into predefined groupes
- Find similar content
- Find associations/patterns in actions/behaviors
- Identify key topics/summarize text
- Detect anomalies/fraud
- Ranking search results
- Many others…

A Few Algorithms

- Collaborative Filtering
- Clustering Techniques (k-means, spectral clustering, …)
- Classification Algorithms (logistic regression, SVM, …)
- Association Rules
- Frequent Pattern Mining
- Others…

Conventional Advanced Analytics Cycle



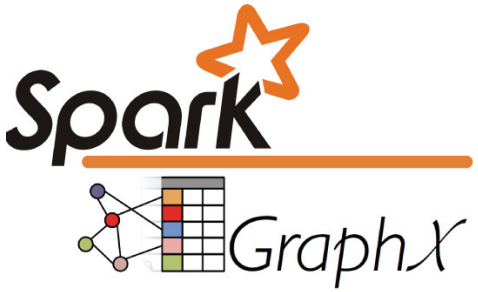
A Few Big Data Analytics Tools

- Spark
- IBM SPSS Modeler
- Microsoft HDInsight
- Dataiku
- Knime
- Talend
- RapidMiner
- Lumify
- Elasticsearch
- R-Programming
- OpenRefine
- Splice Machine
- Plotly
- Skytree
- And so many others…

https://bigdataanalyticsnews.com/top-big-data-analytics-tools/

A Few Tools for Spark

- MLlib Pipelines API
- H2O Sparkling Water
- SparkR
- Mahout
- GraphX
- And so many others…



Interacting with Spark

- Spark's interactive REPL (Read Eval Print Loop environment) shell (in Python or Scala)
  - Try complex commands and/or some code snippets

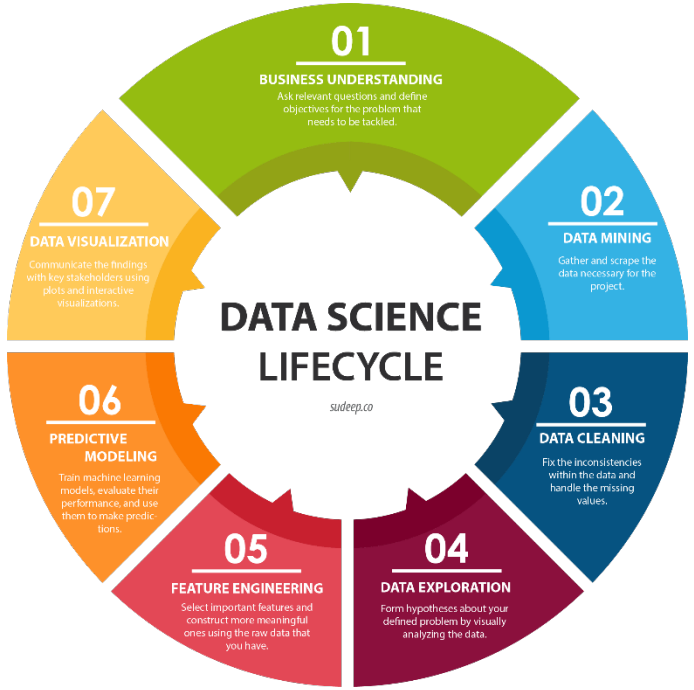
- Spark application (with SBT for example)
  - Develop a full application included in a Java Archive file

- Web-based Notebooks:
  - To present the main results and tell a story
  - Many tools:
    - Zeppelin: A web-based notebook that enables interactive data analytics.
    - Jupyter: Evolved from the IPython Project
    - SparkNotebook: forked from the scala-notebook
    - RStudio: for Spark R

Advanced Analytics Persistence

Data Science

- Prototype
- Create the pipeline
- Evaluate the pipeline



Software Engineering

- Re-implement the pipeline for production (in Java for instance)
- Deploy the pipeline

Advanced Analytics and Big Data

- Ability to learn on large corpus of data is a real boon for advanced analytics
- Even simplistic models shine when they are trained on huge amount of data
- Big Data democratising advanced analytic for general public
- A wide variety of advanced analytics application have started to emerge with big data toolsets

MLlib

Advanced Analytics in Spark

- Spark is the first general-purpose big data processing engine built for advanced analytics from day one

- The initial design in Spark was driven by machine learning optimization
  - Caching - For running on data multiple times
  - Accumulator - To keep state across multiple iterations in memory
  - Good support for CPU intensive tasks with laziness

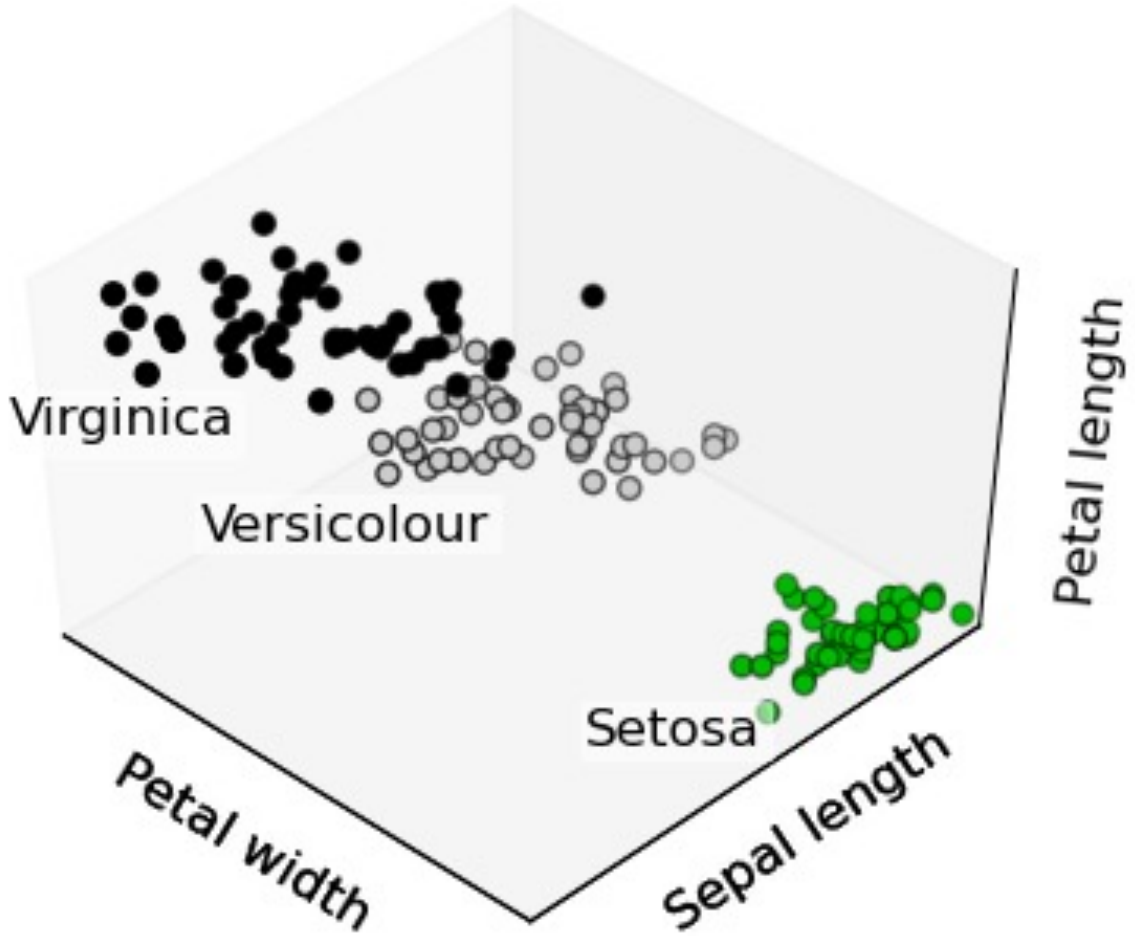
What is MLlib?

- MLlib is a Spark subproject providing machine primitives:
- Initial contribution from AMPLab, UC Berkeley
- Some examples of algorithms:
  - classification: logistic regression, linear support vector machine (SVM), naive Bayes
  - regression: generalized linear regression (GLM)
  - collaborative filtering: alternating least squares (ALS)
  - clustering: k-means
  - decomposition: singular value decomposition (SVD), principal component analysis (PCA)

Why MLlib?

- Scalability
- Performance
- Scala, Java & Python API
- Integration with Spark and its other components (SparkSQL, …)
- Rapid improvements in speed & robustness
- Ongoing development & Large community
- Easy to use, well documented

Clustering: Iris Dataset



Example: K-means in Spark

```
// Load and parse the data.
val data = sc.textFile("kmeans_data.txt")
val parsedData = data.map(_.split(‘ ').map(_.toDouble)).cache()

// Cluster the data into two classes using KMeans.
val clusters = KMeans.train(parsedData, 2, numIterations = 20)

// Compute the sum of squared errors.
val cost = clusters.computeCost(parsedData)
println("Sum of squared errors = " + cost)
```

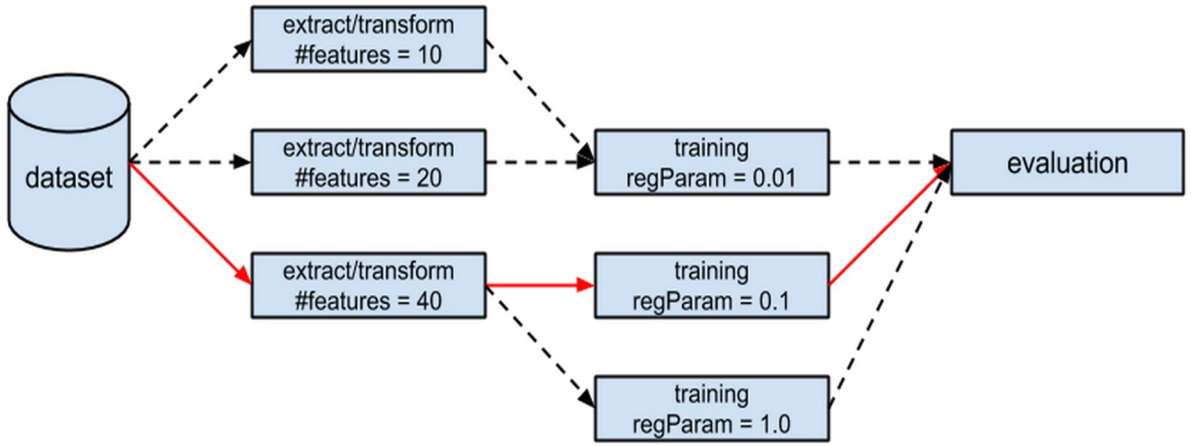
kmeans_data.txt

```
0.0 0.0 0.0
0.1 0.1 0.1
0.2 0.2 0.2
9.0 9.0 9.0
9.1 9.1 9.1
9.2 9.2 9.2
```

MLlib Guide

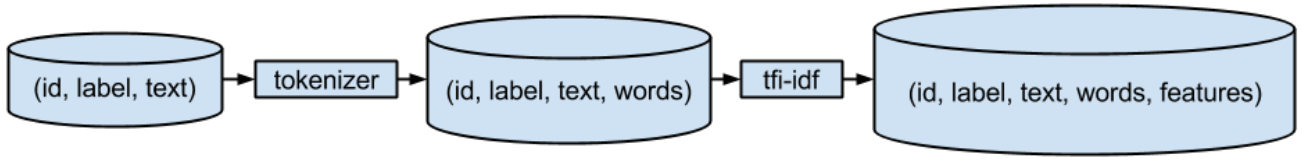
Pipeline API

- Pipeline is a series of algorithms (feature transformation, model fitting, ...)
- Easy workflow construction
- Distribution of parameters into each stage
- Makes MLlib easy to use
- Uses uniform dataset representation - DataFrame from SparkSQL (multiple named columns)



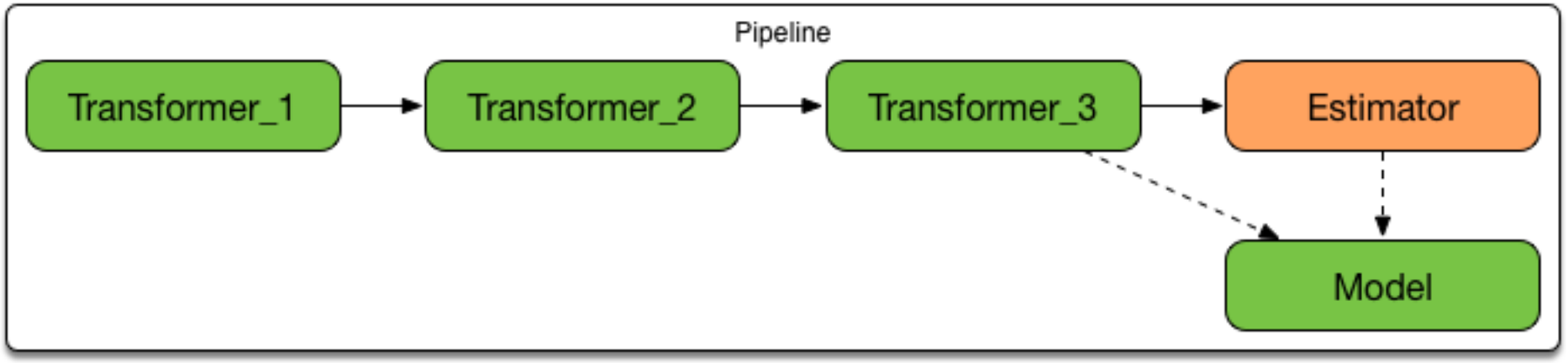
Dataset Abstraction

- A dataset is represented by Spark SQL's dataframe and a processing pipeline by a sequence of dataset transformations.
- MLlib leverages on Spark SQL for several reasons: data import/export, flexible column types and operations, and execution plan.
- Data import/export is the start/end point of the pipeline.
- Feature transformations usually form the majority of a practical pipeline. A feature transformation can be viewed as appending new columns created from existing columns.
- Example:



Pipeline

- A pipeline consists of a sequence of stages.
- There are two basic types of pipeline stages: **Transformer** and **Estimator**.
- A Pipeline specifies a ML workflow.



Main concepts in Pipelines

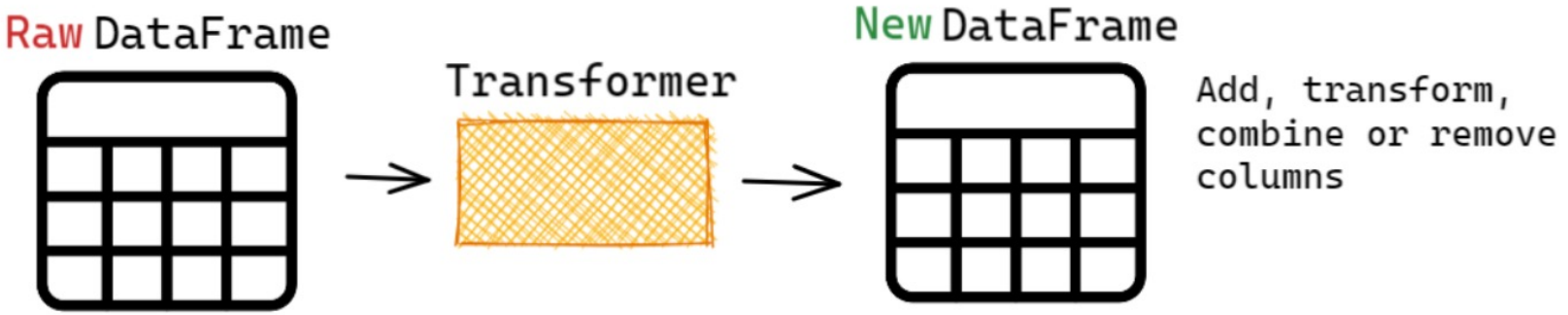
- **DataFrame:** This ML API uses DataFrame from Spark SQL as an ML dataset, which can hold a variety of data types. A DataFrame could have different columns storing text, feature vectors, true labels, and predictions.

- **Transformer:** A Transformer is an algorithm which can transform one DataFrame into another DataFrame. For instance, a ML model is a Transformer which transforms a DataFrame with features into a DataFrame with predictions.

- **Estimator:** An Estimator is an algorithm which can be fit on a DataFrame to produce a Transformer. For instance, a learning algorithm is an Estimator which trains on a DataFrame and produces a model.

- **Parameter:** All Transformers and Estimators share a common API for specifying parameters.

Transformer

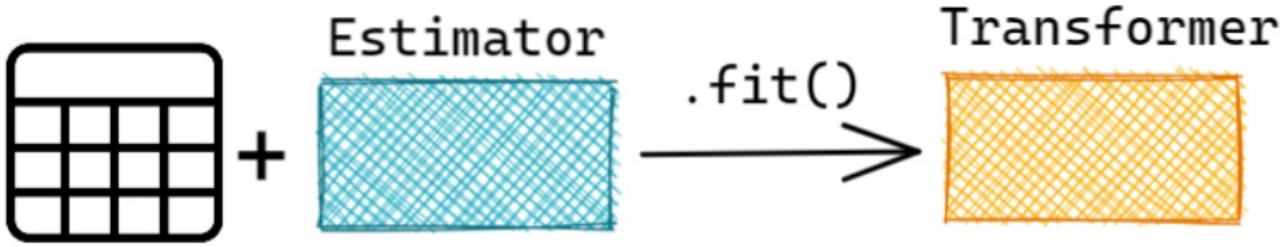


- A Transformer is an abstraction that includes feature transformers and learned models.

- Technically, a Transformer implements a method transform(), which converts one DataFrame into another, generally by appending one or more columns.

- A few examples
  - A tokenizer is a Transformer that transforms a dataset with text into an dataset with tokenized words.
  - A feature transformer might take a DataFrame, read a column (e.g., text), map it into a new column (e.g., feature vectors), and output a new DataFrame with the mapped column appended.
  - A learning model might take a DataFrame, read the column containing feature vectors, predict the label for each feature vector, and output a new DataFrame with predicted labels appended as a column.

Estimator



- An Estimator abstracts the concept of a learning algorithm or any algorithm that fits or trains on data.
- Technically, an Estimator implements a method fit(), which accepts a DataFrame and produces a Model, which is a Transformer.
- For example, a learning algorithm such as LogisticRegression is an Estimator, and calling fit() trains a LogisticRegressionModel, which is a Model and hence a Transformer.
- A Pipeline itself is an Estimator. Thus, after a Pipeline's fit() method runs, it produces a PipelineModel, which is a Transformer.
    - val PipelineModel = pipeline.fit(trainingDataset)
- The PipelineModel is used at test time.

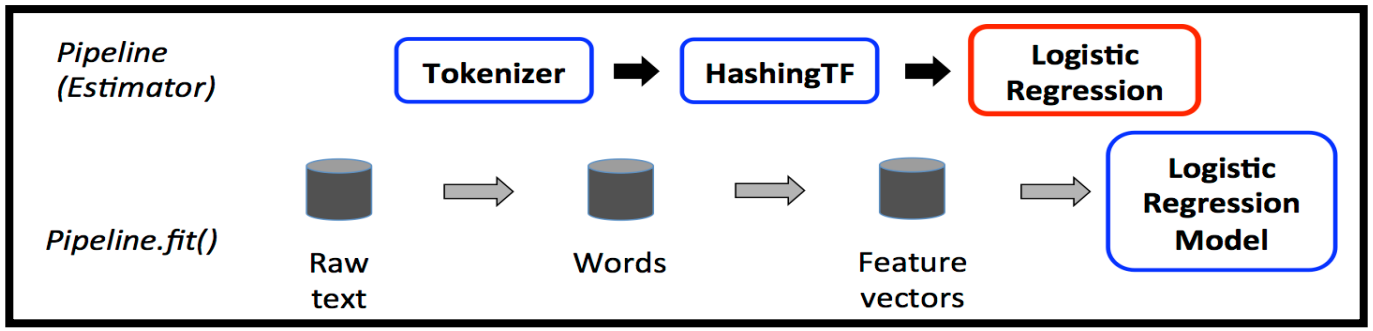
How the pipeline works

- The stages are run in order, and the input DataFrame is transformed as it passes through each stage.

- For Transformer stages, the transform() method is called on the DataFrame.

- For Estimator stages, the fit() method is called to produce a Transformer (which becomes part of the PipelineModel, or fitted Pipeline), and that Transformer's transform() method is called on the DataFrame.

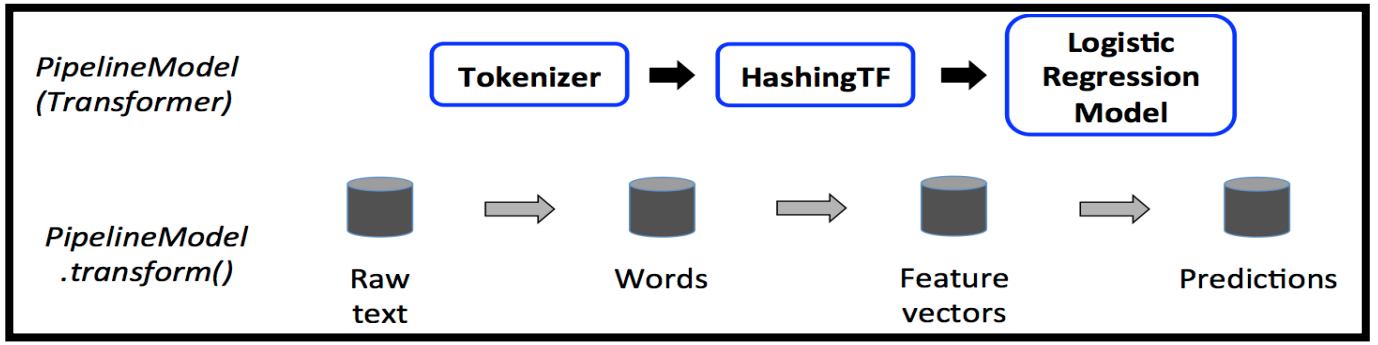
From Estimator to Transformer



- A Pipeline with three stages. The first two (Tokenizer and HashingTF) are Transformers (blue), and the third (LogisticRegression) is an Estimator (red).

- Since LogisticRegression is an Estimator, the Pipeline first calls LogisticRegression.fit() to produce a LogisticRegressionModel, which is a Transformer.

- If the Pipeline had more Estimators, it would call the LogisticRegressionModel's transform() method on the DataFrame before passing the DataFrame to the next stage



Extracting, transforming and selecting features

- **Feature Extractors:**
  - TF-IDF, Word2Vec, CountVectorizer, FeatureHasher
- **Feature Transformers:**
  - Tokenizer, StopWordsRemover, n-gram, Binarizer, PCA, PolynomialExpansion, Discrete Cosine Transform (DCT), StringIndexer, IndexToString, OneHotEncoder, VectorIndexer, Interaction, Normalizer, StandardScaler, RobustScaler, MinMaxScaler, MaxAbsScaler, Bucketizer, ElementwiseProduct, SQLTransformer, VectorAssembler, VectorSizeHint, QuantileDiscretizer, Imputer
- **Feature Selectors:**
  - VectorSlicer, Rformula, ChiSqSelector, UnivariateFeatureSelector, VarianceThresholdSelector
- **Locality Sensitive Hashing:**
  - LSH Operations: Feature Transformation, Approximate Similarity Join, Approximate Nearest Neighbor Search,
  - LSH Algorithms: Bucketed Random Projection for Euclidean Distance, MinHash for Jaccard Distance

Example: VectorAssembler()

```
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.linalg.Vectors

val dataset = spark.createDataFrame(
  Seq((0, 18, 1.0, Vectors.dense(0.0, 10.0, 0.5), 1.0))
).toDF("id", "hour", "mobile", "userFeatures", "clicked")

val assembler = new VectorAssembler()
  .setInputCols(Array("hour", "mobile", "userFeatures"))
  .setOutputCol("features")

val output = assembler.transform(dataset)
```

```
 id | hour | mobile | userFeatures     | clicked
----|------|--------|------------------|---------
 0  | 18   | 1.0    | [0.0, 10.0, 0.5] | 1.0
```

```
 id | hour | mobile | userFeatures     | clicked | features
----|------|--------|------------------|---------|-----------------------------
 0  | 18   | 1.0    | [0.0, 10.0, 0.5] | 1.0     | [18.0, 1.0, 0.0, 10.0, 0.5]
```

Classification and regression

- **Classification**
  - Logistic regression, Decision tree classifier, Random forest classifier, Gradient-boosted tree classifier, Multilayer perceptron classifier, Linear Support Vector Machine, One-vs-Rest classifier, Naive Bayes, Factorization machines classifier
- **Regression**
  - Linear regression, Generalized linear regression, Decision tree regression, Random forest regression, Gradient-boosted tree regression, Survival regression, Isotonic regression, Factorization machines regressor
- **Linear methods**
- **Factorization Machines**
- **Decision trees**
- **Tree Ensembles**
  - Random Forests, Gradient-Boosted Trees (GBTs)

Example: Decision Tree Classification (1/3)

Input Columns

Param name	Type(s)	Default	Description
labelCol	Double	"label"	Label to predict
featuresCol	Vector	"features"	Feature vector

Output Columns

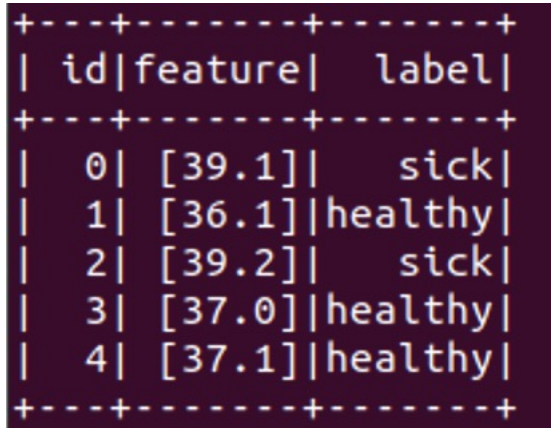
Param name	Type(s)	Default	Description	Notes
predictionCol	Double	"prediction"	Predicted label	
rawPredictionCol	Vector	"rawPrediction"	Vector of length # classes, with the counts of training instance labels at the tree node which makes the prediction	Classification only
probabilityCol	Vector	"probability"	Vector of length # classes equal to rawPrediction normalized to a multinomial distribution	Classification only
varianceCol	Double		The biased sample variance of prediction	Regression only

https://spark.apache.org/docs/latest/ml-classification-regression.html#decision-trees

Example: Decision Tree Classification (2/3)

```
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.feature.{IndexToString,StringIndexer}
import org.apache.spark.ml.classification.DecisionTreeClassifier
import org.apache.spark.ml.linalg.Vectors

// Define the training dataset
val dfTrain = spark.createDataFrame(Seq(
  (0, Vectors.dense(39.1), "sick"),
  (1, Vectors.dense(36.1), "healthy"),
  (2, Vectors.dense(39.2), "sick"),
  (3, Vectors.dense(37.0), "healthy"),
  (4, Vectors.dense(37.1), "healthy")
)).toDF("id", "feature", "label")
```



```
+---+-------+-------+
| id|feature|  label|
+---+-------+-------+
|  0| [39.1]|   sick|
|  1| [36.1]|healthy|
|  2| [39.2]|   sick|
|  3| [37.0]|healthy|
|  4| [37.1]|healthy|
+---+-------+-------+
```

Example: Decision Tree Classification (3/3)

id	feature	label
0	[39.1]	sick
1	[36.1]	healthy
2	[39.2]	sick
3	[37.0]	healthy
4	[37.1]	healthy

```
// Convert original labels into indexed labels
val labelIndexer = new StringIndexer().setInputCol("label").setOutputCol("indexedLabel").fit(dfTrain)

// Convert indexed labels back to original labels.
val labelConverter = new IndexToString().setInputCol("prediction").setOutputCol("predictedLabel").setLabels(labelIndexer.labelsArray(0))

// Train a DecisionTree model.
val dt = new DecisionTreeClassifier().setLabelCol("indexedLabel").setFeaturesCol("feature")

// Chain labelIndexer, tree and labelConverter in a Pipeline.
val pipeline = new Pipeline().setStages(Array(labelIndexer, dt, labelConverter))

// Train model.
val model = pipeline.fit(dfTrain)

// Make predictions.
val predictions = model.transform(dfTrain)
```

id	feature	label	indexedLabel	rawPrediction	probability	prediction	predictedLabel
0	[39.1]	sick	1.0	[0.0,2.0]	[0.0,1.0]	1.0	sick
1	[36.1]	healthy	0.0	[3.0,0.0]	[1.0,0.0]	0.0	healthy
2	[39.2]	sick	1.0	[0.0,2.0]	[0.0,1.0]	1.0	sick
3	[37.0]	healthy	0.0	[3.0,0.0]	[1.0,0.0]	0.0	healthy
4	[37.1]	healthy	0.0	[3.0,0.0]	[1.0,0.0]	0.0	healthy

Clustering

- **K-means**
- **Latent Dirichlet allocation (LDA)**
- **Bisecting k-means**
- **Gaussian Mixture Model (GMM)**
- **Power Iteration Clustering (PIC)**

ML Tuning: model selection and hyperparameter tuning

- **Model selection (a.k.a. hyperparameter tuning)**
- **Cross-Validation**
- **Train-Validation Split**

Conclusion

Conclusion

- Analytics is multidisciplinary
- Recommend action or guide decision making rooted in business context
- Advanced analytics is generally coming with Big Data tools
- Many tools already exists and regularly come to light