**Big Data**

The definition changes from company to company; here are some thoughts some large companies share on Big Data definitions:

* Is the **derivation of value** from traditional relational database-driven business decision making, **augmented** with **new sources**of unstructured data. (Oracle)
* **Opportunities** emerge in organizations generating a median of 300 terabytes of data a week. The **most common forms** of data analyzed in this way are business transactions stored in relational databases, followed by documents, e-mail, sensor data, blogs, and social media. (Intel)
* The term increasingly used to describe the**process** of **applying** serious computing power— latest in **machine learning** and artificial intelligence— to seriously **massive** and often highly complex **sets** of information. (Microsoft)
* Is **data** which “exceed(s) the capacity or capability of current or conventional methods and systems.” In other words, the notion of “big” is relative to the current standard of computation. (National Institute of Standards and Technology)

Big data is exciting; it can change policy decisions and the way we do day-to-day business. But to harness these benefits, we need to address several challenges first. Organizations have a long tradition of capturing transactional data. Apart from that, organizations nowadays are capturing additional data from its operational environment at an increasingly fast speed.

#### ****Examples of Big Data****

### [**Web Data**](https://northeastern.instructure.com/courses/67866/pages/lesson-1-3-introduction-to-a-big-data?module_item_id=5689072)

Customer level web behavior data such as page views, searches, reading reviews, and purchasing can be captured. They can enhance performance in areas such as next best offer, churn modeling, customer segmentation, and targeted advertisement.

### [**Text Data**](https://northeastern.instructure.com/courses/67866/pages/lesson-1-3-introduction-to-a-big-data?module_item_id=5689072)

E-mail, news, Facebook feeds, documents, etc. are the biggest and most widely applicable types of big data. The focus is typically on extracting key facts from the text and then use the facts as inputs to other analytic processes (for example, automatically classify insurance claims as fraudulent or not).

### [**Time and Location Data**](https://northeastern.instructure.com/courses/67866/pages/lesson-1-3-introduction-to-a-big-data?module_item_id=5689072)

GPS and mobile phones as well as Wi-Fi connection make time and location information a growing source of data. At an individual level, many organizations come to realize the power of knowing when their customers are at which location.

Equally important is to look at time and location data at an aggregated level. As more individuals open up their time and location data more publicly, lots of interesting applications start to emerge. Time and location data is one of the most privacy-sensitive types of big data and should be treated with great caution.

### [**Smart Grid and Sensor Data**](https://northeastern.instructure.com/courses/67866/pages/lesson-1-3-introduction-to-a-big-data?module_item_id=5689072)

Sensor data are collected nowadays from cars, oil pipes, windmill turbines, and they are collected at an extremely high frequency. Sensor data provides powerful information on the performance of engines and machinery. It enables the diagnosis of problems more easily and faster development of mitigation procedures.

**Big Data vs Traditional Data**

**First:**

**Big Data**can be an entirely new source of data.

* For instance, most of us have experience with **online shopping**. The transactions we execute are not fundamentally different transactions from what we would have done traditionally.
* An organization may **capture web transactions**, but they are really just more of the same transactions that have been captured for years (e.g. purchasing records).
* However, actually **capturing browsing behavior** (how do you navigate on the site, for instance) as customers execute a transaction creates fundamentally new data.

**Second:**

Sometimes one can argue that the **speed** of **data** feed has **increased** to such an extent that it qualifies as a new data source.

* For instance, your power meter has probably been read manually each month for years. Now we have a smart meter that automatically reads it every 10 minutes.
* One can argue that it is the **same data**. It can also be argued that the frequency is so high now that it enables a very different, more in-depth level of analytics that such data is really a new data source.

**Third:**

**Increasingly** more semi-structured and unstructured data are coming in.

* Most **traditional data sources** are in the structured realm.
* **Structured data** are the ones like the receipts from your grocery store, the data on your salary slip, accounting information on the spreadsheet, and pretty much everything that can fit nicely in a relational database.
* Every piece of information included is **known ahead**of **time**, comes in a specified format and occurs in a specified order; this makes it easy to work with.

### [**Volume**](https://northeastern.instructure.com/courses/67866/pages/lesson-1-4-four-vs-in-big-data?module_item_id=5689074)

Volume in big data refers to a large amount of data that you have to deal with. Nowadays, data is produced in a very large quantity.

Think about surveillance cameras installed in a major city, such as Boston, LA, or New York. The number of these cameras might be in the thousands, and each of them is providing a constant video stream, resulting in massive amounts of data, even within one day.

### [**Velocity**](https://northeastern.instructure.com/courses/67866/pages/lesson-1-4-four-vs-in-big-data?module_item_id=5689074)

Velocity refers to the speed at which the data arrives.

Again, if we consider surveillance cameras, they provide data at constant speed and often at high resolution. This is a lot of data at high speeds. The internet also provides a vast amount of data at very high speed. A company's firewall system has to monitor the high-speed data trying to enter their network.

In the context of cyber security, it's crucial to deal with this data of high velocity and to make sure that it's not a cyber attack. Due to the high velocity of data, it might not be feasible to store or check all of the data. To deal with this issue, we look at sampling techniques that store a representative fraction of the data.

### [**Veracity**](https://northeastern.instructure.com/courses/67866/pages/lesson-1-4-four-vs-in-big-data?module_item_id=5689074)

Veracity refers to the uncertainty that comes with data.

Often, data is not complete and can be noisy. So you cannot completely rely on all aspects of the data that arrives, and you have to deal with abnormalities of the data. Think of location services on phones. If every user provides their location, then this location is usually not precise, but within a range of, let's say, 100 yards. The data may not be complete, as the GPS coordinates cannot be obtained at some locations.

Dealing with this data often requires a data-cleaning process that reduces veracity. While this can remove abnormalities, it often still leaves you with incomplete data, and you would need to fill in the blanks when using it for your application.

### [**Variety**](https://northeastern.instructure.com/courses/67866/pages/lesson-1-4-four-vs-in-big-data?module_item_id=5689074)

The variety of big data refers to the different sources of data.

Data can come in various forms - images, videos, audio, sensor data, and so on. For a specific application, you might have to integrate data from various sources. Often the data provided is unstructured and doesn't arrive in a coordinated way. Then, you have to rely on your different data sources.

To learn more about the Four V’s in Big Data, please see the interactive model below.

**Social Media**

There are millions of people using Facebook and Twitter.

* All the data is produced in an online fashion arriving in the form of a data stream. Users post a variety of data online on Facebook, such as text, images, videos. Similarly, Twitter has short text messages.
* The data is high volume and arrives with high velocity at the Facebook/Twitter servers.
* Users may be tagged by their location using GPS coordinates.
* These coordinates are usually imprecise leading to veracity of the data.

**Fraud Detection in Banking Transactions**

* Banking produces millions of transactions per day. These transactions have to be processed safely and reliably. Thinking about a bank’s transactions over a month results in a vast volume of data.
* Fraud detection refers to finding bogus transactions that have been triggered by criminals. This can be by using a stolen credit card or even only its details. You see that for fraud detection you would have to deal with large volumes of data, each transaction arriving rapidly, and a decision having to be made as soon as a transaction arrives.
* There are some indicators that can be used to identify fraud, for example a credit card used at an ATM in one country when all other transactions in the previous 2 days have been in another country.
* Finding frauds is hard and the information used to stop a transaction is usually not 100% reliable. You might even have observed this yourself when you tried to use your credit card in a different country and the card was rejected although you were the legitimate user of the card.

**Online Stores**

* Online stores such as Amazon have millions of potential customers that buy a large variety of items from their online servers. These customers produce a very large number of transactions within a short time period.
* Mining these transactions to extract useful information (for example to optimise advertising) has to deal with the large number of users and the variety of items that they have bought. Making a recommendation to one particular user takes into account what the user has bought so far.
* The knowledge gathered about a customer is incomplete and a recommendation system has to rely on the imprecise information that it can obtain from the transaction data of customers and their behaviour on the online store page.

Traditionally, data are stored in relational databases (for example a CRM system for customer data, a supply chain management software for vendor related information).

Some of these data are extracted periodically from the operational database, transformed and loaded into a data warehouse for reporting and further analysis. This is typically in the realm of BI (Business Intelligence). Such a process and tool set fall short when dealing with big data.

An example:

One of the largest publicly discussed Hadoop cluster (Yahoo’s) was at 455 petabytes in 2014 and it's grown since then. There simply are no parallel relational databases or data warehouses that have come even close to those kinds of numbers.

Another sweet spot for Hadoop (over relational technology) is when data comes in an unstructured format:

* Audio
* Video
* Text

It is worthwhile to mention that there is a general misconception that new technology, such as **Hadoop** is replacing other technologies, such as relational databases. **It is not the case**.

It is more likely that they are being added alongside each other. The sweet spot for a massive, parallel relational platform for instance, is dealing with high-value transactional data that is already structured.  The platform needs to support a large amount of user and applications that ask repeated questions of known data (where a fixed schema and optimization pays off) with enterprise level security and performance guarantee.

It is often called the **Hadoop eco-system** when discussing the various lays of technologies used to deal with big data.

For a complete list, please refer to [The Hadoop Ecosystem Table (Links to an external site.)](https://hadoopecosystemtable.github.io/).

For instance, stack might look like:

* Amazon web service for infrastructure (in the Cloud and pay as you go);
* Apache HDFS (Hadoop Distributed File System) for distributed file system;
* MapReduce or Spark for distributed programming model;
* Cassandra or HBase for non-relational distributed database management system;
* Hive for execute SQL on top of Hadoop;
* Mahout for Machine learning library and math library, on top of MapReduce;
* R for data analytics and visualization;

**Analytical Techniques**

Most of the widely used analytical techniques falls into one of the following categories:

* Statistical methods:
  + forecasting
  + regression analysis
* Database querying
* Data warehouse
* Machine learning and data mining

**Visualization**

When analysis is done, the results need to be communicated to various stakeholders. One of the hardest parts of an analysis is producing quality supporting graphics. Conversely, a good graph is one of the best ways to present findings.

Graphs are used primarily for two reasons:

* exploratory data analysis
* presenting results

### **Lesson 2-1 — New Paradigm for Big Data**

In the past decade the amount of data being created has skyrocketed. More than 30,000 gigabytes of data are generated every second, and the rate of data creation is only accelerating.

**The data we deal with is diverse:**

* Blog Posts
* Tweets
* Social Network Interactions
* Photos
* Audio Video
* Log messages from servers
* Scientific measurements

This astonishing growth in data has profoundly affected businesses.

When traditional systems cannot handle the high volume of transitions, then the company might lose business.

For example, during Thanksgiving of  2019, Costco's website went down most of its customers couldn't place any orders. ([https://www.cnbc.com/2019/11/29/costcos-website-goes-down-on-thanksgiving-others-see-outages.html (Links to an external site.)](https://www.cnbc.com/2019/11/29/costcos-website-goes-down-on-thanksgiving-others-see-outages.html))

However, being that Costo is a member-only site, they were able to extend online Thanksgiving deals for additional 2 days.

#### ****From Traditional Data to Big Data****

Traditional database systems, such as relational databases, have been pushed to the limit. In an increasing number of cases these systems are breaking under the pressures of “Big Data.” Traditional systems, and the data management techniques associated with them, and are failing to scale to Big Data. To tackle the challenges of Big Data, a new breed of technologies has emerged.

Many of these new technologies have been grouped under the term NoSQL. In some ways, these new technologies are more complex than traditional databases, and in other ways they’re simpler.

These systems can scale to vastly larger sets of data, but using these technologies effectively requires a fundamentally new set of techniques. They aren’t one-size-fits-all solutions.

**Many of these Big Data systems were pioneered by Google:**

* Distributed file systems
* MapReduce computation framework
* Distributed locking services

**Another notable pioneer in the space was Amazon**

* Innovative distributed key/value store called Dynamo

**The open source community responded in the years following with:**

* Hadoop
* Hbase
* MongoDB
* Cassandra
* RabbitMQ

**How does traditional database technology hit the limits? Simple project:**

* Build a simple web analytics application
* Application should track the number of page views for any URL that the customer wants to track
* The customer's web page pings the application web server every time a page view is received
* Application should tell, at any point, what the top 100 URLs are by the number of pages

### **Lesson 2-2 — Traditional Data Problems**

Click on the interactive graphic below to learn more.  The arrow on the top right will reveal three bullet points and seven steps.

[Traditional Data Problems Graphic Text Description](https://northeastern.instructure.com/courses/67866/files/8818584?wrap=1)

### [**Consider this scenario:**](https://northeastern.instructure.com/courses/67866/pages/lesson-2-2-traditional-data-problems?module_item_id=5689098)

You do some Google searches for how to scale a write-heavy relational database.

You find that the best approach is to use multiple database servers and spread the table across all the servers. Each server will have a subset of the data for the table. This is known as horizontal partitioning or sharding.

This technique spreads the write load across multiple machines. The sharding technique you use is to choose the shard for each key by taking the hash of the key modded by the number of shards.

You write a script to map over all the rows in your single database instance, and split the data into four shards (partitions).

[**Your application code needs to know how to find the shard for each key.**](https://northeastern.instructure.com/courses/67866/pages/lesson-2-2-traditional-data-problems?module_item_id=5689098)

* You wrap a library around your database-handling code that reads the number of shards from a configuration file, and you redeploy all of your application code.
* You have to modify your top-100-URLs query to get the top 100 URLs from each shard and merge those together for the global top 100 URLs.
* As the application gets more and more popular, you keep having to “reshard” the database into more shards to keep up with the write load.

Each time gets more and more painful because there’s so much more work to coordinate. You have to do all the “resharding” in parallel and manage many active worker scripts at once.

If you forget to update the application code with the new number of shards, and it causes many of the increments to be written to the wrong shards. So you have to write a one-off script to manually go through the data and move whatever was misplaced.

Eventually you have so many shards that it becomes a not-infrequent occurrence for the disk on one of the database machines to go bad. That portion of the data is unavailable while that machine is down.

You update your queue/worker system to put increments for unavailable shards on a separate “pending” queue that you attempt to flush once every five minutes. You use the database’s replication capabilities to add a worker node to each shard so you have a backup in case the main node goes down.

While working on the queue/worker code, you accidentally deploy a bug to production that increments the number of pageviews by two, instead of by one, for every URL.

You don’t notice until 24 hours later, but by then the damage is done. Your weekly backups don’t help because there’s no way of knowing which data got corrupted. After all this work trying to make your system scalable and tolerant of machine failures, your system has no resilience to a human making a mistake.

As the simple web analytics application evolved, the system continued to get more and more complex: queues, shards, replicas, resharding scripts, and so on.

Developing applications on the data requires a lot more than just knowing the database schema. Your code needs to know how to talk to the right shards, and if you make a mistake, there’s nothing preventing you from reading from or writing to the wrong shard.

#### ****Traditional DB – What Went Wrong?****

* The traditional database is not self-aware of its distributed nature, so it can’t help you deal with shards, replication, and distributed queries.
* All that complexity got pushed to you both in operating the database and developing the application code.
* The system is not engineered for human mistakes.
* The system keeps getting more and more complex, making it more and more likely that a mistake will be made.

### **Lesson 2-3 — Big Data Techniques**

The Big Data techniques address the scalability and complexity issues in a dramatic fashion. The databases and computation systems you use for Big Data are aware of their distributed nature by design.

Instead of storing the pageview counts as your core dataset, which you continuously mutate as new page views come in, you store the raw pageview information. That raw pageview information is never modified.

When you make a mistake, you might write bad data, but at least you won’t destroy good data.

#### ****Big Data Architecture****

The tools here are a combination of traditional systems (like RDBMS, WebServer, etc) as well as new open source Hadoop tools like HDFS, YARN, MapReduce, HBase etc.  As such, these tools on their own are not a panacea. When intelligently used in conjunction with one another, you can produce scalable systems for arbitrary data problems with human-fault tolerance and a minimum of complexity.

This is called the Lambda Architecture. The properties we expect from Big Data systems are as much about complexity as they are about scalability. Not only must a Big Data system perform well and be resource efficient, it must be easy to reason about as well. What this means is that the new big data systems shouldn't be difficult to add additional machines, configure them, apply new software fixes or upgrade to higher versions of software used.  Adding new machines and pathing software are generally a bit of complex steps in traditional systems.

**Desired properties of a Big Data system:**

### [**Robustness and Fault Tolerance**](https://northeastern.instructure.com/courses/67866/pages/lesson-2-3-big-data-techniques?module_item_id=5689100)

* Systems need to behave correctly despite:
  + Machines going down randomly
  + The complex semantics of consistency in distributed databases
  + Duplicated data
  + Concurrency
* Avoid Complexities
  + Systems has to be resilient to human error by providing a clear and simple mechanism for recovery

### [**Low Latency Reads and Updates**](https://northeastern.instructure.com/courses/67866/pages/lesson-2-3-big-data-techniques?module_item_id=5689100)

The vast majority of applications require reads to be satisfied with very low latency. The update latency requirements vary a great deal between applications. Some applications require updates to propagate immediately, but in other applications a latency of a few hours is fine. Low latency reads and updates must be achieved without compromising the robustness of the system.

### [**Scalability**](https://northeastern.instructure.com/courses/67866/pages/lesson-2-3-big-data-techniques?module_item_id=5689100)

Scalability is the ability to maintain performance in the face of increasing data or load by adding resources to the system.

The Lambda Architecture is horizontally scalable across all layers of the system stack: scaling is accomplished by adding more machines.

### [**Generalization**](https://northeastern.instructure.com/courses/67866/pages/lesson-2-3-big-data-techniques?module_item_id=5689100)

A general system can support a wide range of applications. Because the Lambda Architecture is based on functions of all data, it generalizes to all applications:

* Financial management systems
* Social media analytics
* Scientific applications
* Social networking

### [**Extensibility**](https://northeastern.instructure.com/courses/67866/pages/lesson-2-3-big-data-techniques?module_item_id=5689100)

Extensible systems allow functionality to be added with a minimal development cost. Part of making a system extensible is making it easy to do large-scale migrations. Being able to do big migrations quickly and easily is core to the approach of system extensibility.

### [**Ad Hoc Queries**](https://northeastern.instructure.com/courses/67866/pages/lesson-2-3-big-data-techniques?module_item_id=5689100)

Being able to do ad hoc queries on your data is extremely important. Nearly every large dataset has unanticipated value within it. Being able to mine a dataset arbitrarily gives opportunities for business optimization and new applications.

### [**Minimal Maintenance**](https://northeastern.instructure.com/courses/67866/pages/lesson-2-3-big-data-techniques?module_item_id=5689100)

Maintenance is the work required to keep a system running smoothly:

* Add machines to scale
* Keep processes up and running
* Debugging anything that goes wrong in production

Combat implementation complexity by relying on simple algorithms and simple components. The more complex a system, the more likely something will go wrong. The Lambda Architecture pushes complexity out of the core components and into pieces of the system whose outputs are discardable after a few hours.

### [**Debuggability**](https://northeastern.instructure.com/courses/67866/pages/lesson-2-3-big-data-techniques?module_item_id=5689100)

A Big Data system must provide the information necessary to debug the system when things go wrong. The key is to be able to trace, for each value in the system, exactly what caused it to have that value. “Debuggability” is accomplished in the Lambda Architecture through the functional nature of the batch layer and by preferring to use recomputation algorithms when possible.

### **Lesson 2-4 —Lambda Architecture**

Nathan Marz came up with the term Lambda Architecture (LA) for a generic, scalable and fault-tolerant data processing architecture, based on his experience working on distributed data processing systems at Backtype and Twitter.

The Lambda Architecture aims to satisfy the needs for a robust system that is fault-tolerant, both against hardware failures and human mistakes, being able to serve a wide range of workloads and use cases, and in which low-latency reads and updates are required.

The resulting system should be linearly scalable, and it should scale out rather than up. The main idea of the Lambda Architecture is to build Big Data systems as a series of layers.

[Lambda Architecture Graphic Text Description](https://northeastern.instructure.com/courses/67866/files/8818582?wrap=1)

### [**Batch Layer**](https://northeastern.instructure.com/courses/67866/pages/lesson-2-4-lambda-architecture?module_item_id=5689102)

The portion of the Lambda Architecture that implements the batch view = function(all data) equation is called the batch layer.

The batch layer stores the master copy of the dataset and precomputes batch views on that master dataset.

The batch layer needs to be able to do two things:

* Store an immutable, constantly growing master dataset
* Compute arbitrary functions on that dataset

The batch layer runs in a while(true) loop and continuously recomputes the batch views from scratch. Hadoop is the canonical example of a batch-processing system.

### [**Serving Layer**](https://northeastern.instructure.com/courses/67866/pages/lesson-2-4-lambda-architecture?module_item_id=5689102)

Serving layer is a specialized distributed database that loads in a batch view and makes it possible to do random reads on it.

When new batch views are available, the serving layer automatically swaps those in so that more up-to-date results are available. A serving layer database supports batch updates and random reads. It doesn’t need to support random writes which makes it extremely simple.

ElephantDB, the serving layer database is only a few thousand lines of code.

### [**Speed Layer**](https://northeastern.instructure.com/courses/67866/pages/lesson-2-4-lambda-architecture?module_item_id=5689102)

Speed layer computes arbitrary functions on arbitrary data in real time to compensate for those last few hours of serving layer updates.

Its goal is to ensure new data is represented in query functions as quickly as needed for the application requirements. Speed layer only looks at most recent data, whereas the batch layer looks at all the data at once.

In order to achieve the smallest latencies possible, the speed layer doesn’t look at all the new data at once.

It updates the real-time views as it receives new data instead of recomputing the views from scratch like the batch layer does.

### **Lesson 2-5 — Recent Trends in Technology**

#### ****CPU Aren't Getting Faster****

We’ve started to hit the physical limits of how fast a single CPU can go. To scale to more data we have to parallelize computations.

We have seen the rise of shared-nothing parallel algorithms and their corresponding systems, such as MapReduce. Instead of scaling by buying a better machine (vertical scaling), systems scale by adding more machines (horizontal scaling).

#### ****Elastic Clouds****

We see the rise of elastic clouds, also known as Infrastructure as a Service (Amazon Web Services). Elastic clouds let you increase or decrease the size of your cluster nearly instantaneously, so if you have a big job you want to run, you can allocate the hardware temporarily. Elastic clouds dramatically simplify system administration.

#### ****Vibrant Open Source Ecosystem for Big Data****

* Batch computation systems - Hadoop project:
  + Hadoop Distributed File System (HDFS) - distributed, fault- tolerant storage system that can scale to petabytes of data
  + Hadoop MapReduce - horizontally scalable computation framework that integrates with HDFS
* Serialization frameworks - provide tools and libraries for using objects between languages. They can serialize an object into a byte array from any language, and then deserialize that byte array into an object in any language.
* Random-access NoSQL databases - Cassandra, HBase, MongoDB, Voldemort, Riak, CouchDB, and others. They sacrifice the full expressiveness of SQL and instead specialize in certain kinds of operations.
* Messaging/queuing systems - provide a way to send and consume messages between processes in a fault-tolerant and asynchronous manner. Message queue is a key component for doing real-time processing.(Apache Kafka)
* Real-time computation systems - high throughput, low latency, stream-processing systems.(Storm)

#### ****We've Got the Data, What now?****

Depending on the type of data scale, different data quality and preprocessing techniques can be used.

The quality of the models, charts and studies in data analytics depends on the quality of the data being used.

The nature of the application domain, human error, the integration of different data sets and the methodology used to collect data can generate data sets that are noisy, inconsistent, or contain duplicate records.

Even with a large number of robust descriptive and predictive algorithms available to deal with noisy, incomplete, inconsistent or redundant data, an increasing number of real applications have their findings harmed by poor-quality data.

### **Lesson 2-6 — Data Quality**

In data sets collected directly from storage systems (actual data), it is estimated that noise can represent 5% or more of the total data set. When these data are used by algorithms that learn from data (ML algorithms) - the analysis problem can look more complex than it really is if there is no data pre-processing.

This increases the time required for the induction of assumptions or models, resulting in models that do not capture the true patterns present in the data set. The elimination or even just the reduction of these problems can lead to an improvement in the quality of knowledge extracted by data analysis processes.

Data quality is important and can be affected by internal and external factors:

**External factors** are related to faults in the data collection process, and can involve the absence of values for some attributes and the voluntary or involuntary addition of errors to others.

**Internal factors** can be linked to the measurement process and the collection of information through the attributes chosen.

### **Lesson 2-7 — Data Quality Main Problems**

There are four main issues in Data Quality. They present distinct problems and each have their own solution approach. Please see the problems in more detail below.

### [**Missing Values**](https://northeastern.instructure.com/courses/67866/pages/lesson-2-7-data-quality-main-problems?module_item_id=5689108)

In real-life applications, it is common that some predictive attribute values for some of the records may be missing in the data set.

There are several causes of missing values, among them:

* Attributes values only recorded for some time after start of collection
* The value of an attribute being unknown at time of collection
* Distraction, misunderstanding or refusal at time of collection
* Attribute not required for particular objects
* Non-existence of a value
* Fault in the data collection device
* Cost or difficulty of assigning a class label to an object in classification problems.

Since many data analysis techniques were not designed to deal with a data set with missing values, the data set must be pre-processed.

Several approaches can be used:

**Ignore missing values**

* Use for each object only the attributes with values, without paying attention to missing values
* Modify a learning algorithm to allow it to accept and work with missing values

**Remove objects**

* Use only those objects with values for all attributes.

**Make estimates**

* Fill the missing values with estimates based on values for this attribute in the other objects.

### [**Inconsistency**](https://northeastern.instructure.com/courses/67866/pages/lesson-2-7-data-quality-main-problems?module_item_id=5689108)

A data set can also have inconsistent values. The presence of inconsistent values in a data set usually reduces the quality of the model induced by ML algorithms.

Inconsistent values can be found in the predictive and/or target attributes. An example of an inconsistent value in a predictive attribute is a zip code that does not match the city name. This inconsistency can be due to a mistake or fraud.

A good policy to deal with inconsistent values in the predictive attribute is to treat them as missing values.

### [**Redundancy**](https://northeastern.instructure.com/courses/67866/pages/lesson-2-7-data-quality-main-problems?module_item_id=5689108)

Redundant objects are those that do not bring any new information to a data set. Objects very similar to other objects represent an irrelevant data.

Redundancy occurs mainly in the whole set of attributes. Redundant data could be due to small mistakes or noise in the data collection (same addresses for people whose names differ by just a single letter).

Redundant data can be duplicate data. Deduplication is a preprocessing technique whose goal is to identify and remove copies of objects in a data set.

### [**Noise**](https://northeastern.instructure.com/courses/67866/pages/lesson-2-7-data-quality-main-problems?module_item_id=5689108)

Noisy data are data that do not meet the set of standards expected for them.

Noise can be caused by incorrect or distorted measurements, human error or even contamination of the samples. Noise detection can be performed by adaptation of classification algorithms or by the use of noise filters for data preprocessing.

[Two Graphs - Data With and Without Noise Image Text Description](https://northeastern.instructure.com/courses/67866/files/8818503?wrap=1)

### [**Outlier**](https://northeastern.instructure.com/courses/67866/pages/lesson-2-7-data-quality-main-problems?module_item_id=5689108)

In a data set, outliers are anomalous values or objects. They can also be defined as objects whose values for one or more predictive attributes are very different from the values found in the same predictive attributes of other objects.

In contrast to noisy data points, outliers can be legitimate values. There are several data analysis applications whose main goal is to find outliers in a data set. Particularly in anomaly detection tasks, the presence of outliers can indicate the presence of noise.

### **Lesson 2-8 — Data Pre-Processing: Converting to a Different Scale Type**

There are four strategies in Pre-Processing

1. Converting to a Different Scale Type
2. Converting to a Different Scale
3. Data Transformation
4. Dimensionality Reduction

#### ****Converting to a Different Scale Type****

### [**Converting to a Different Scale Type – Nominal to Relative**](https://northeastern.instructure.com/courses/67866/pages/lesson-2-8-data-pre-processing-converting-to-a-different-scale-type?module_item_id=5689109)

Some ML algorithms can use only data of a particular scale type. Data can be converted from a qualitative scale to a quantitative one.

**Original Data to Converted Data shown below.**

**Original Data**

| **Food** | **Age** | **Distance** | **Company** |
| --- | --- | --- | --- |
| Chinese | 51 | Close | Good |
| Italian | 43 | Very close | Good |
| Italian | 82 | Close | Good |
| Burgers | 23 | Far | Bad |
| Chinese | 46 | Very far | Good |
| Chinese | 29 | Too far | Bad |
| Burgers | 42 | Very far | Good |
| Chinese | 38 | Close | Bad |
| Italian | 31 | Far | Good |

**Converted Data**

| **F1** | **F2** | **F3** | **Age** | **Distance** | **Company** |
| --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 51 | Close | Good |
| 0 | 1 | 0 | 43 | Very close | Good |
| 0 | 1 | 0 | 82 | Close | Good |
| 1 | 0 | 0 | 23 | Far | Bad |
| 0 | 0 | 1 | 46 | Very far | Good |
| 0 | 0 | 1 | 29 | Too far | Bad |
| 1 | 0 | 0 | 42 | Very far | Good |
| 0 | 0 | 1 | 38 | Close | Bad |
| 0 | 1 | 0 | 31 | Far | Good |

### [**Converting to a Different Scale Type – Ordinal to Relative or Absolute**](https://northeastern.instructure.com/courses/67866/pages/lesson-2-8-data-pre-processing-converting-to-a-different-scale-type?module_item_id=5689109)

For ordinal values, the conversion is more intuitive, since we can convert to natural numbers, starting with the value 0 for the smallest value and, for each subsequent value, adding 1 to the previous value.

| **Nominal** | **Natural Number** | **Gary Code** | **Thermometer Code** |
| --- | --- | --- | --- |
| Small | 0 | 00 | 000 |
| Italian | 1 | 01 | 001 |
| Italian | 2 | 10 | 011 |
| Burgers | 3 | 11 | 111 |

Gray code keeps the distance between two consecutive values as a different value in one of the binary values.

The thermometer code starts with a binary vector with only 0 values and substitutes one 0 value by 1, from right to left, as the ordinal value increases.

### [**Converting to a Different Scale Type – Relative or Absolute to Ordinal or Nominal**](https://northeastern.instructure.com/courses/67866/pages/lesson-2-8-data-pre-processing-converting-to-a-different-scale-type?module_item_id=5689109)

Discretization is a process to convert quantitative values to nominal or ordinal ones.

Discretization used to reduce the number of quantitative values.

Discretization has two steps:

1. Define the number of quantitative values - bins.
2. Define the interval of values to associate with each bin

There are at least two ways to define how to associate quantitative values to a different bins:

1. Association by width
2. Association by frequency

| **Quantitative values** | **Conversion by width** | **Conversion by frequency** |
| --- | --- | --- |
| 2 | A | A |
| 3 | A | A |
| 5 | A | A |
| 7 | A | B |
| 10 | B | B |
| 15 | B | B |
| 16 | C | C |
| 19 | C | C |
| 20 | C | C |

* Association by width - the intervals will have the same range: the same difference between the largest and smallest values:
  + 2-8, 8-15, 16-22 (the difference is 6)
* Association by frequency - each interval will have the same number of values:
  + 2-5, 7-15, 16-20

Quantitative scale does not necessarily have to have all possible values. Upper and lower limits of the interval do not need to be in the data set and that some values, which will never appear, can be left out of the intervals.

| **Quantitative values** | **Conversion by width** | **Conversion by frequency** |
| --- | --- | --- |
| 2 | A | A |
| 3 | A | A |
| 5 | A | A |
| 7 | A | B |
| 10 | B | B |
| 15 | B | B |
| 16 | C | C |
| 19 | C | C |
| 20 | C | C |

### **Lesson 2-9 — Data Pre-Processing: Converting to a Different Scale**

Converting data in a scale to another scale of the same type is necessary in several situations, such as when using distance measures.

This kind of conversion is typically done in order to have different attributes expressed on the same scale; a process known as “normalization”. The normalization is carried out for each attribute individually.

There are two ways to normalize the data:

1. Min–Max Rescaling
2. Standardization

#### ****Converting to a Different Scale – Min-Max Rescaling****

Min–max rescaling, converts numerical values to values in a given interval.

To convert a set of values to values in the interval [0.0, ..., 1.0], you simply subtract the smallest value from each of the values in the set and divide the new value by the amplitude: the difference between the maximum and minimum values.

#### ****Converting to a Different Scale – Standardization****

When using standardization, first subtract the average of the attribute values and then divide the result by the standard deviation of these values. As a result, the standardized values of the attribute will have now an average of 0.0 and standard deviation of 1.0.

### **Lesson 2-10 — Data Pre-Processing: Data Transformation**

[Data Pre-Processing Graphic Text Description](https://northeastern.instructure.com/courses/67866/files/8818580?wrap=1)

Another important issue for data summarization is transformations that might be necessary to perform in order to simplify the analysis or allow the use of particular modeling techniques.

**Applying a logarithmic function to the values of a predictive attribute:**

Usually performed for skewed distributions, when some of the values are much larger (or much smaller) than the others. The logarithm makes the distribution less skewed. Thus, log transformations make the interpretation of highly skewed data easier.

**Conversion to absolute values:**

For some predictive attributes, the value’s magnitude is more important than its sign, if the value is positive or negative.

### **Lesson 2-11 — Data Pre-Processing: Dimensionality Reduction**

**The dimensionality reduction of a data set can bring several benefits:**

* Reduces the training time, decreases memory needed and improves performance of ML algorithms
* Eliminates irrelevant attributes and reduces the number of noisy attributes
* Allows the induction of simpler and more easily interpretable models
* Makes the data visualization easier to understand and allows visualization of data sets with a high number of attributes
* Reduces the cost of feature extraction, making ML-based Technologies accessible to a larger number of people

There are two alternatives to reduce the number of attributes: attribute aggregation or attribute selection

#### ****Dimensionality Reduction - Attribute Aggregation****

In attribute aggregation, we replace a group of attributes with a new attribute: a combination of the attributes in the group. Attribute aggregation, also known as multidimensional scaling, reduces the data to a given number of attributes, allowing an easier visualization. Attribute aggregation techniques project the original data set into a new, lower-dimensional space, but keeping the relevant information.

Several techniques have been proposed for attribute aggregation in the literature.

* Principal Component Analysis – PCA
* Independent Component Analysis – ICA
* Multidimensional Scaling – MDS

Some of the techniques work by linearly combining the original attributes, creating a smaller number of attributes, referred to as a set of components. Other techniques can also create non-linear combinations of the original attributes of a data set.

A key concept to understand aggregation techniques is the concept of data projection. A projection transforms a set of attributes in one space to a set of attributes in another space. The original data are named sources and the projected data signals. A simple projection would be to remove some attributes from the original data set, creating a new data, the projected data set, with the remaining attributes. In a projection, we want to keep as much of the information present in the original set of attributes as possible. Ideally, the projection removes redundancy and noise from the original data.

Instead of aggregating attributes, another approach to reduce dimensionality is by selecting a subset of the attributes. This approach can speed up the learning process, since a smaller number of operations will need to be made. Attribute selection techniques can be roughly divided into three categories:

1. Filters
2. Wrappers
3. Embedded

### [**Filters**](https://northeastern.instructure.com/courses/67866/pages/lesson-2-11-data-pre-processing-dimensionality-reduction?module_item_id=5689114)

Filters look for relations between the predictive attribute values and the target attribute. Filters rank the attributes according to this relation. If the values of a predictive attribute have a strong relation with a label attribute value, this predictive attribute receives a high ranking position.

### [**Wrappers**](https://northeastern.instructure.com/courses/67866/pages/lesson-2-11-data-pre-processing-dimensionality-reduction?module_item_id=5689114)

Wrappers explicitly use a classifier to guide the attribute selection process. Wrappers select the set of predictive attributes that provides the highest predictive performance for the classifier, regardless of the attribute ranking positions.

Instead of ranking the predictive attributes, wrappers usually look for the subset of attributes with the best predictive ability. Wrappers capture the relationship between attributes and reduce the chance of selecting redundant attributes.

### [**Embedded**](https://northeastern.instructure.com/courses/67866/pages/lesson-2-11-data-pre-processing-dimensionality-reduction?module_item_id=5689114)

In the embedded category, attribute selection is performed as an internal procedure of a predictive algorithm. One predictive technique able to perform embedded attribute selection is a decision tree induction algorithm. When these algorithms are applied to a data set, they produce a predictive model and select a subset of predictive attributes chosen to be the most relevant for classification model induction.

### **Module Overview**

We will examine different big data analysis processing methods, and try some of them in lab exercises. We will walk through the setup, configuration, and usage by reviewing several examples. The knowledge you gain from this module will provide you with options for working on big data sets in your future projects.

### **Learning Objectives**

By the end of this module, you should be able to:

* Assess the use of big data storage and computing technologies using big data analytics applications.
* Evaluate how Big Data solutions that utilize data lakes to store unstructured data.
* Use methodologies to analyze big data sets and determine insights, including the use of Apache Spark technology.

### **Reading Resources**

* [Understanding Sentiment Analysis: What It Is & Why It's Used (Links to an external site.)](https://www.brandwatch.com/blog/understanding-sentiment-analysis/) from Krisian Bannister from Brandwatch.  
  This article discusses the uses of sentiment analysis in analyzing big data.
* [What's the Difference between Supervised and Unsupervised Learning? (Links to an external site.)](https://dataconomy.com/2015/01/whats-the-difference-between-supervised-and-unsupervised-learning/) by Eileen McNulty from Dataconomy.
* This article explains the difference between supervised and unsupervised learning with an example.
* [Supervised Machine Learning: Classification (Links to an external site.)](https://builtin.com/data-science/supervised-machine-learning-classification) by Badreesh Shetty.  
  This article provides explanation of various methods of classification.
* [Multivariate Testing vs. A/B Testing (Links to an external site.)](https://www.optimizely.com/optimization-glossary/multivariate-test-vs-ab-test/) from Optimizely.  
  This article explains the difference between Multivariate and A/B Testing and when to use each.
* [Why Graph Databases Are so Effective in Big Data Analytics (Links to an external site.)](https://www.cleverism.com/graph-databases-effective-big-data-analytics/) from Cleverism.  
  This article discusses what graphs are, where they are used and why they are used in Big Data analytics.

### **Practical Resources**

* [Big Data Methodologies.](https://northeastern.instructure.com/courses/67866/files/9759605?wrap=1)
* [Top 28 Cheat Sheets for Machine Learning, Data Science, Probability, SQL & Big Data (Links to an external site.)](https://www.analyticsvidhya.com/blog/2017/02/top-28-cheat-sheets-for-machine-learning-data-science-probability-sql-big-data/)
  + The cheat sheet list segregated separately for each of the following topics:
    - Machine Learning.
    - Data Science.
    - Probability.
    - SQL.
    - Big Data.
  + There are cheat sheets on tools & techniques, various libraries & languages.

### **Task List**

* Review Module 3 content.
* Review Required Learning Resources.
* Participate in the Module 3 Discussion.
* Complete and submit Module 3 Assignment before the due date.

### **Lesson 3-1 — Big Data and Data Science**

**Big Data, a technology for data processing, is now defined by the “four Vs”:**

* **Volume:** how to store big data (data repositories for large amounts of data)?
* **Variety:** how to put together data from different sources?
* **Velocity:** how to deal with data arriving very fast, in streams known as data streams?
* **Veracity:** how to deal with noise and abnormality in data? Is the data that is being stored, and mined meaningful to the problem being analyzed and current to make the right decision?

Another term used as a synonym for big data is data science.

Big data are data sets that are too large to be managed by conventional data-processing technologies, requiring the development of new techniques and tools for data storage, processing and transmission.

**Examples of these tools include:**

* MapReduce
* Hadoop
* Spark
* Storm

**The Data Volume is not the only characterization of big data. The word “big” can refer to:**

* The number of data sources
* The importance of the data
* The need for new processing techniques
* How fast data arrive
* The combination of different sets of data so they can be analyzed in real time

**Big data's major concern is technology.**

The technology provides a computing environment, not only for analytics, but also for other data processing tasks. Data science is concerned with the creation of models able to extract patterns from complex data and the use of these models in real-life problems.

Data science extracts meaningful and useful knowledge from data, with the support of suitable technologies. Data science has a close relationship to analytics and data mining. Data science goes beyond data mining by providing a knowledge extraction framework, including statistics and visualization.

Therefore big data collects and data science discovers.

As data increases in size, velocity and variety, new computer technologies become necessary. These new technologies, which include hardware and software, must be easily expanded as more data are processed. This property is known as scalability.

One way to obtain scalability is by distributing the data processing tasks into several computers, which can be combined into clusters of computers. Clusters produced by analytics techniques (not computer clusters) is where a data set is partitioned in groups within it.

### **Lesson 3-2 — MapReduce**

[Methods Used in Big Data Video Transcript](https://northeastern.instructure.com/courses/67866/files/8818541?wrap=1)

[Actions](https://northeastern.instructure.com/courses/67866/pages/lesson-3-2-mapreduce?module_item_id=5689132)

Even if processing power is expanded by combining several computers in a cluster, creating a distributed system, conventional software for distributed systems cannot keep up with big data. One of the limitations is the efficient distribution of data among the different processing and storage units.

To deal with these requirements, new software tools and techniques have been developed. One of the first techniques developed for big data processing using clusters was MapReduce.

MapReduce is a programming model that consists of two steps: map and reduce. The most famous implementation of MapReduce is called Hadoop.

#### ****MapReduce****

[Data to Destination Process Image Text Description](https://northeastern.instructure.com/courses/67866/files/8818561?wrap=1)

[Actions](https://northeastern.instructure.com/courses/67866/pages/lesson-3-2-mapreduce?module_item_id=5689132)

#### ****Big Data System Requirements****

To efficiently solve a big data problem, a distributed system must adhere to the following requirements:

* Ensure that no chunk of data is lost and the whole task is concluded. (If one or more computers has a failure, their tasks, and the corresponding data chunk, must be assumed by another computer in the cluster.)
* Repeat the same task, and corresponding data chunk, in more than one cluster computer (redundancy). (If one or more computers fail, the redundant computer carries on with the task.)
* Failed computers can return to the cluster when they are fixed.
* Computers can be easily removed from the cluster or extra ones included in it as the processing demand changes.

#### ****MapReduce Example****

MapReduce divides the data set into parts (chunks) and stores it in the memory of each cluster computer.

As an example:

* Suppose that you need to calculate the average salary of 1 billion people and you have a cluster with 1000 computers, each with a processing unit and a storage memory.
* The people can be divided into 1000 chunks (subsets) with data from 1 million people each.
* Each chunk can be processed independently by one of the computers.
* The results (1000) produced by each of these computers (the average salary of 1 million people) can be averaged, returning the final salary average.

To be characterized as small data, a data set must have a size that allows its full understanding by a user. When these data are collected to be stored and processed in large data servers they become big data.

#### ****Small Data****

[Small Data Image Text Description](https://northeastern.instructure.com/courses/67866/files/8818508?wrap=1)

[Actions](https://northeastern.instructure.com/courses/67866/pages/lesson-3-2-mapreduce?module_item_id=5689132)

### **Lesson 3-3 — Lambda Architecture**

### **Lambda Architecture designed to build Big Data systems using series of layers.**

[Lamda Architecture Image Text Description](https://northeastern.instructure.com/courses/67866/files/8818582?wrap=1)

[Actions](https://northeastern.instructure.com/courses/67866/pages/lesson-3-3-lambda-architecture?module_item_id=5689134)

#### ****Data Model for Big Data****

[Data Model for Big Data Graphic Text Description](https://northeastern.instructure.com/courses/67866/files/8818565?wrap=1)

[Actions](https://northeastern.instructure.com/courses/67866/pages/lesson-3-3-lambda-architecture?module_item_id=5689134)

### **Lesson 3-4 — The Properties of Big Data**

***Information*** is the general collection of knowledge relevant to your Big Data system.

**Data**refers to the information that can’t be derived from anything else. Data serves as the axioms from which everything else is derived.

**Queries** are questions you ask of your data. (query financial transaction history to determine your current bank account balance).

**Views**are information that has been derived from your base data. They are built to assist with answering specific types of queries.

**An example of FaceData social network system (your own FaceBook**)

There are three possible options for storing friendship information for the Social Network System. Each option can be derived from the one to its left, but it’s a one-way process.

[Example of FaceData Network System Graphic Text Description](https://northeastern.instructure.com/courses/67866/files/8818590?wrap=1)

[Actions](https://northeastern.instructure.com/courses/67866/pages/lesson-3-4-the-properties-of-big-data?module_item_id=5689136)

#### ****Why Do We Need Performance?****

Prior to the Big Data era the term Data Warehousing was widely used. The key difference between the two is that Data Warehousing is just one part of the Big Data System, whose primary responsibility is performance.

The idea is to pre-calculate the data to serve the most common queries faster – create views in advance. The natural human behavior is to expect most current data immediately. People have no patience waiting for what is here as of today.

At the same time, people naturally would understand that if historical data needed to be extracted, it should take more time and there is more patience there.

### **Lesson 3-5 — Stock Status Report Example**

The most common example from the business side of Big Data Counting and Management is a Stock Status report.

Let’s say, a company buys and sells a small number of parts (part numbers), but in large volumes. Each transaction is stored for each part with the date.

* Receiving transaction recorded with “+” quantity amount
* Selling transaction recorded with “-” quantity amount

### [**Transactions History**](https://northeastern.instructure.com/courses/67866/pages/lesson-3-5-stock-status-report-example?module_item_id=5689138)

| **Date** | **Part Number** | **Qty** |
| --- | --- | --- |
| 01/05/2016 | Part A | +1,500 |
| 02/09/2016 | Part A | -350 |
| 02/09/2016 | Part B | +25,000 |
| 02/10/2016 | Part C | +15,000 |
| 02/11/2016 | Part B | -5,000 |
| 03/01/2016 | Part C | +12,000 |
| 03/01/2016 | Part B | -5,500 |
| 03/02/2016 | Part C | -10,000 |
| 03/03/2016 | Part A | +12,000 |
| 03/04/2016 | Part B | -1,000 |

**To calculate the current stock status report, we use a simple formula:**

For each Part Number, calculate the *summary* or *quantity*.

As a result, we have a table where each part is only shown once.

The quantity field shows the total quantity of this part we currently have in stock.

It runs every day as the decision has to be made to spend more money, buying more parts or not.

### [**Stock Status As of Now**](https://northeastern.instructure.com/courses/67866/pages/lesson-3-5-stock-status-report-example?module_item_id=5689138)

| **Part Number** | **Current Qty** |
| --- | --- |
| Part A | 13,150 |
| Part B | 13,500 |
| Part C | 17,000 |

The system is designed to calculate this report on demand using all transactions designed to fail.

We can keep adding more hardware resources: RAM, CPUs, etc., but after some period of time, we would need more to keep up with the performance and deliver this report on demand.

No one wants to wait just to see what is going on now.

Complaints from all departments start to accumulate saying: it is faster to go to the warehouse and manually count it, rather than wait for the system to show the data.

This is where a **batch layer** or **data warehouse** changes the game.

1. Batch: New transaction added for Part A“as-is”
2. Batch: Part A quantity updated by subtracting 1,000

### [**Transactions History**](https://northeastern.instructure.com/courses/67866/pages/lesson-3-5-stock-status-report-example?module_item_id=5689138)

| **Date** | **Part Number** | **Qty** |
| --- | --- | --- |
| 01/05/2016 | Part A | +1,500 |
| 02/09/2016 | Part A | -350 |
| 02/09/2016 | Part B | +25,000 |
| 02/10/2016 | Part C | +15,000 |
| 02/11/2016 | Part B | -5,000 |
| 03/01/2016 | Part C | +12,000 |

### [**Stock Status**](https://northeastern.instructure.com/courses/67866/pages/lesson-3-5-stock-status-report-example?module_item_id=5689138)

| **Part Number** | **Current Qty** |
| --- | --- |
| Part A | 13,150 |
| Part B | 13,500 |
| Part C | 16,000 |

New Data 01/06/2018, Part A, -1,000

Stock Status report (query) now runs from the Stock Status table (view) and does not require millions of calculations to be performed each time a report is requested. Stock Status and Transaction History tables (views) are updated instantly with each new data coming in. A new record is always added to the Transaction History. A new record is added to the Stock Status table if there is no matching Part Number exists. An existing record is updated in the Stock Status table for the part that matches the transaction part.

### **Lesson 3-6 — Stock Status Report Example (Continued)**

We’ve covered a task of having today’s stock status data available instantly instead of waiting for processing of millions transactions recorded over the years.

How about yesterday's Stock Status? Should we still wait all this time and process all recorded transactions to see what was the status as of yesterday?

Well, yes, we need to calculate the “in” and “out” quantities to show accurate results. We also might make a decision to have a Stock Status Table for each day. So in 5 years we will have at least 1825 tables (views).

We still can find a compromise between storing “everything” and calculating on demand. One of the most popular solutions is instead calculating parts quantities “forward” from the beginning of the transaction history, calculating them “backward”.

We start from the current stock status table (view) and “reverse” all transactions happening between today and yesterday. The stock status as of beginning of yesterday will only include transaction for two days: today and yesterday, which is significantly smaller number then number of transactions for the last 5 years.

There are hybrid solutions that can be implemented as well.

For example, creating a stock status table for the beginning and for the end of each year. We will have only 10 tables for 5 years. When a Stock Status report request, the Stock Status table (view) representing the Year of the request will be used.

If the date is closer to the beginning of the year, we use the table for the beginning of this year and “add” all transactions between the first date of the year and requested stock status report date.

If the date is closer to the end of the year, we use the table for the end of the year and “reverse” transactions within this range.

### **Lesson 3-8 — The Data is Raw**

A data system answers questions about information you’ve acquired in the past.

When designing your Big Data system, you want to be able to answer as many questions as possible. The master data set is more valuable than views because views can be regenerated from the master data set data but not the other way around. We call this property rawness of data.

Although the concept is straightforward, it’s not always clear what information you should store as your raw data.

#### ****Unstructured vs Normalized****

The unstructured data is rawer than normalized data. When deciding what raw data to store, a common hazy area is the line between parsing and semantic normalization. Semantic normalization is the process of reshaping freeform information into a structured form of data.

#### ****Unstructured vs Normalized****

[Structured Data Image Text Description](https://northeastern.instructure.com/courses/67866/files/8818563?wrap=1)

[Actions](https://northeastern.instructure.com/courses/67866/pages/lesson-3-8-the-data-is-raw?module_item_id=5689144)

Semantic normalization of unstructured location responses to city, state, and country. A simple algorithm will normalize “North Beach” to NULL if it doesn’t recognize it as a San Francisco neighborhood. If you come across a form of data such as an unstructured location string, should you store the unstructured string or the semantically normalized form?

If you store the unstructured string, you can renormalize that data at a later time when you have improved your algorithms. If your algorithm for extracting the data is simple and accurate you should store the results of that algorithm.

If the algorithm is subject to change, due to improvements or broadening the requirements, store the unstructured form of the data. More information does not necessarily mean rawer data.

Let’s say that Tom is a blogger, and he wants to add his posts to his FaceSpace profile. What exactly should you store once Tom provides the URL of his blog?

Storing the pure text of the blog entries is certainly a possibility. But any phrases in italics, boldface, or large font were deliberately emphasized by Tom and could prove useful in text analysis. Store the full HTML of Tom’s blog as your data will require more space: color schemes, stylesheets, JavaScripts, etc.

### **Lesson 3-9 — Data is Immutable**

Immutable data may seem like a strange concept if you’re well-versed in relational databases. In the relational database world, update is one of the fundamental operations.

For immutable data, you don’t update or delete data, you only add more. By using an immutable schema for Big Data systems, you gain two vital advantages:

* Human-fault tolerance
* Simplicity

#### ****Human-fault tolerance****

Human-fault tolerance is an essential property of data systems. People will make mistakes, and you must limit the impact of such mistakes and have mechanisms for recovering from them.

With a mutable data model, a mistake can cause data to be lost, because values are actually overridden in the database. With an immutable data model, no data can be lost. If “bad” data is written - earlier (good) data units still exist.

Fixing the data system is just a matter of deleting the bad data units and re-computing the views built from the master dataset.

#### ****Simplicity****

Mutable data models imply that the data must be indexed in some way so that specific data objects can be retrieved and updated. In contrast, with an immutable data model you only need the ability to append new data units to the master dataset. This doesn’t require an index for your data, which is a huge simplification. Storing a master dataset is as simple as using flat files.

A mutable schema for Social Network user information. The advantages of keeping your data immutable become evident when comparing with a mutable schema.

|  | **Name** | **Age** | **Gender** | **Employer** | **Location** |
| --- | --- | --- | --- | --- | --- |
| 1 | Alice | 25 | Female | Apple | Atlanta, GA |
| 2 | Bob | 36 | Male | SAS | Chicago, IL |
| 3 | Tom | 28 | Male | Google | San Francisco, CA |
| 4 | Charlie | 25 | Male | Microsoft |  |
| ... | ... | ... | ... | ... | ... |

When details change - say, Tom moves to Los Angeles - previous values are overwritten and lost.

An immutable schema for Social Network user information.

* Each field of user information is kept separately.
* Each record is time stamped when it is stored.

### [**Name Data**](https://northeastern.instructure.com/courses/67866/pages/lesson-3-9-data-is-immutable?module_item_id=5689146)

| **id** | **Name** | **Timestamp** |
| --- | --- | --- |
| 1 | Alice | 09/04/2018  07:30:01 |
| 2 | Bob | 10/11/2017  12:04:13 |
| 3 | Tom | 05/08/2018  18:01:15 |
| 4 | Charlie | 09/12/2016  01:12:01 |

### [**Age Data**](https://northeastern.instructure.com/courses/67866/pages/lesson-3-9-data-is-immutable?module_item_id=5689146)

| **id** | **Age** | **Timestamp** |
| --- | --- | --- |
| 1 | 25 | 09/04/2018  07:30:01 |
| 2 | 36 | 10/11/2017  12:04:13 |
| 3 | 28 | 05/08/2018  18:01:15 |
| 4 | 25 | 09/12/2016  01:12:01 |

### [**Location Data**](https://northeastern.instructure.com/courses/67866/pages/lesson-3-9-data-is-immutable?module_item_id=5689146)

| **id** | **Location** | **Timestamp** |
| --- | --- | --- |
| 1 | Atlanta, GA | 09/04/2018  07:30:01 |
| 2 | Chicago, IL | 10/11/2017  12:04:13 |
| 3 | San Francisco, CA | 05/08/2018  18:01:15 |
| 4 | Washington, DC | 09/12/2016  01:12:01 |

Each field is tracked in a separate table, and each row has a timestamp for when it’s known to be true.

Rather than storing a current snapshot of the world, as done by the mutable schema, immutable one creates a separate record every time a user’s information evolves.

Accomplishing this requires two changes.

1. Each field of user information is tracked in a separate table.
2. Each unit of data is tied to a moment in time when the information is known to be true.

When Tom moved to Los Angeles on June 17, 2018, a new record was added to the location table, timestamped by when he changed his profile. Now there are two location records for Tom (user ID #3), and because the data units are tied to particular times, they can both be true. Tom’s current location involves a simple query on the data: look at all the locations, and pick the one with the most recent timestamp.

By keeping each field in a separate table, you only record the information that changed. This requires less space for storage and guarantees that each record is new information and is not simply carried over from the last record.

### [**Location Data**](https://northeastern.instructure.com/courses/67866/pages/lesson-3-9-data-is-immutable?module_item_id=5689146)

| **id** | **Location** | **Timestamp** |
| --- | --- | --- |
| 1 | Atlanta, GA | 09/04/2018  07:30:01 |
| 2 | Chicago, IL | 10/11/2017  12:04:13 |
| 3 | San Francisco, CA | 05/08/2018  18:01:15 |
| 4 | Washington, DC | 09/12/2016  01:12:01 |
| 5 | Los Angeles | 06/17/2018  15:30:05 |

Initial Data about Tom location - GREEN

Additional record about new Tom location - BLUE

#### ****Data is Eternally True****

The key consequence of immutability is that each piece of data is true in perpetuity. Each piece of data, once true, must always be true.

Immutability wouldn’t make sense without this property, this is why each piece of data tagged with a timestamp as a practical way to make data eternally true. In general, your master dataset consistently grows by adding new immutable and eternally true pieces of data.

There are some special cases, though, in which data deleted and these cases are not incompatible with data being eternally true:

Garbage collection:

* Implemented to delete all data units that have low value
* Used to implement data retention policies that control the growth of the master dataset. (keep only one location per person per year)

Regulations

* Government regulations may require purging data from databases under certain conditions

In both cases, deleting the data is not a statement about the truthfulness of the data but instead, it’s a statement about the value of the data.

### **Lesson 3-10 — The Fact-Based Model**

Collective data, such as Tom’s friend list in the figure, are represented as multiple, independent facts.

**The Fact-Based Model**

[Fact Based Model Image Text Description](https://northeastern.instructure.com/courses/67866/files/8818505?wrap=1)

[Actions](https://northeastern.instructure.com/courses/67866/pages/lesson-3-10-the-fact-based-model?module_item_id=5689148)

Identifiability- facts should be associated with a uniquely identifiable piece of data. If facts come for the same time we might encounter the same exact data record.

If we encounter two identical pageview records, there’s no way to tell whether they refer to two distinct events or if a duplicate entry was accidentally introduced into our dataset.

To distinguish different pageviews, you can add a nonce to your schema - a 64-bit number randomly generated for each pageview. The addition of the nonce makes it possible to distinguish pageview events from each other, and if two pageview data units are identical (all fields, including the nonce), you know they refer to the exact same event.

Making facts identifiable means that we can write the same fact to the master dataset multiple times without changing the semantics of the master dataset. Queries can filter out the duplicate facts when doing their computations.

Having distinguishable facts makes implementing the rest of the Lambda Architecture much easier.

### [**Benefits of the fact-based model:**](https://northeastern.instructure.com/courses/67866/pages/lesson-3-10-the-fact-based-model?module_item_id=5689148)

**The Dataset is Queryable at Any Time in its History**

* Instead of storing only the current state of the world, we have the ability to query the data for any time covered by the dataset.
* This is a direct consequence of facts being timestamped and immutable.
* “Updates” and “Deletes” are performed by adding new facts with more recent timestamps, but because no data is actually removed, we can reconstruct the state of the world at the time specified by the query.

**The Data is Human-Fault Tolerant**

* Human-fault tolerance is achieved by simply deleting any erroneous facts.
* Suppose you mistakenly stored that Tom moved from San Francisco to Los Angeles.
* By removing the Los Angeles fact, Tom’s location is automatically “reset” because the San Francisco fact becomes the most recent information.

**The Dataset Easily Handles Partial Information**

* Storing one fact per record makes it easy to handle partial information about an entity without introducing NULL values into your dataset.
* Suppose Tom provided his age and gender but not his location or profession.
* Dataset would only have facts for the known information - any “absent” fact would be logically equivalent to NULL.
* Additional information that Tom provides at a later time would naturally be introduced via new facts.

**The Data Storage and Query Processing Layers are Separate**

* Another key advantage of the fact-based model that is in part due to the structure of the Lambda Architecture itself.
* By storing the information at both the batch and serving layers, we have the benefit of keeping your data in both normalized and denormalized forms and reaping the benefits of both.
* Data normalization is completely unrelated to the semantic normalization term, it refers to storing data in a structured manner to minimize redundancy and promote consistency.

The mutually exclusive choice between normalized and denormalized schemas is necessary because, for relational databases, queries are performed directly on the data at the storage level.

The objectives of query processing and data storage are cleanly separated in the Lambda Architecture. In the Lambda Architecture, the master dataset is fully normalized and no data is stored redundantly.

Updates are easily handled because adding a new fact with a current timestamp “overrides” any previous related facts.

### **Lesson 3-11 — Graph Schemas**

Each fact within a fact-based model captures a single piece of information. The facts alone don’t convey the structure behind the data. There is no description of the types of facts contained in the dataset, nor any explanation of the relationships between them.

Graph schemas capture the structure of a dataset stored using the fact-based model. Graph schema representing the relationships between the facts. Graph provides a useful visualization of existing facts and the relationship between them.

[Core Components of a Graph Schema Graphic Text Description](https://northeastern.instructure.com/courses/67866/files/8818586?wrap=1)

[Actions](https://northeastern.instructure.com/courses/67866/pages/lesson-3-11-graph-schemas?module_item_id=5689149)