School Of Computer And Communication Sciences

Applied Data Analysis
Summary

Dr. Catasta Michele
Distributed Information Systems Laboratory (LSIR)
michele.catasta@epfl.ch

June 10, 2016

Contents

1 Introduction

1.1 General information about the course

This course covers multiple topics in the data science field such as **Data Wrangling**, **Data Management**, **Data Mining**, **Machine Learning**, **Visualization**, **Statistics** and **Story telling**. It's about **breadth**, not depth. Indeed, Data science is evolving really quickly, hence learning in depth a specific tool won't pay off.

1.2 Data Science

When we talk about Data Science, we often use the term Big Data as the enormous amount of data that exist in the world. But Big Data is not only about collecting huge amount of data. It is challenging but not enough. The real value comes from the insights. The *internet* companies (Google, Facebook, etc.) understood this many years ago.

An accurate definition of Data Analysis is given by Wikipedia:

> **Analysis of data** *is a process of* **inspecting**, **cleaning**, **transforming**, *and* **modeling data** *with the goal of* **discovering useful information**, *suggesting conclusions, and supporting decision-making. Data analysis has multiple facets and approaches, encompassing diverse techniques under a variety of names,* **in different business**, *science, and social science* **domains**. Wikipedia - Data Analysis

Therefore, a Data Scientist has to master different kind of skills such as **Mathematics** (for the Statistics), **Programming** and the **Domain Expertise**. Drew Conway's Venn diagram, Figure 1, shows the different combination man can obtain with these three skills.

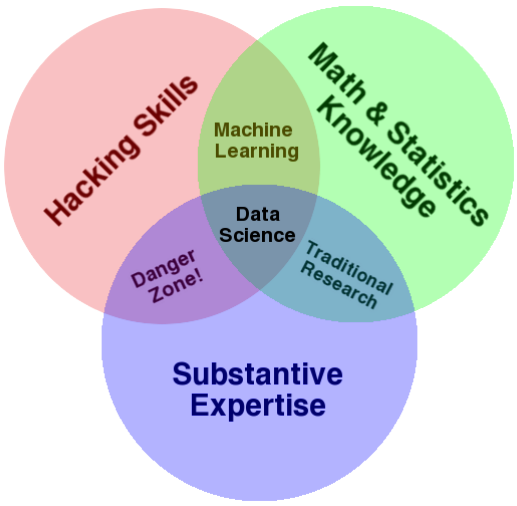


Figure 1: Venn Diagram describing the different combination of skills used by a Data Scientist (by Drew Conway)

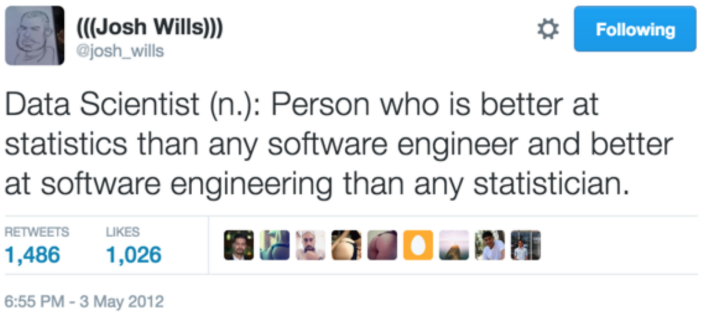
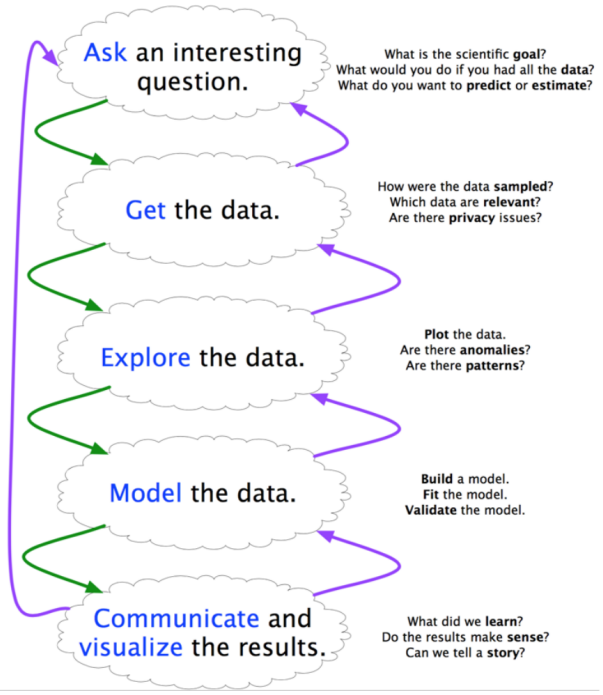


Figure 2: A tweet from Josh Wills, Data Scientist at Slack.

A practical definition of Data Science

Data Science is about the whole processing pipeline to extract information out of data. As such, a Data Scientist **understands and cares about the whole data pipeline**.



A **data pipeline** consists of 3 steps:

1. Preparing to run a model.
   Gathering, cleaning, integrating, restructuring, transforming, loading, filtering, deleting, combining, merging, verifying, extracting, shaping
2. Running the model
3. Communicating the results

A "good" Data Scientist will always go back and forth between the steps. The diagram on the left shows exactly what can happen.

In this course, you will develop the following skills:

data muning/scraping/sampling/cleaning in order to get an informative, manageable data setlength

data storage and management in order to be able to access data quickly and reliably during subsequent analysis

exploratory data analysis to generate hypotheses and intuition about the data

prediction based on statistical tools such as regression, classification, and clustering

communication of results through visualization, stories and interpretable summaries

2 Basic concepts

A data science student is attended to understand the **Grammar of Data Science**. Having some backgrounds in SQL concepts is also a good thing because, as it is very common, people loves to make example with it. Here is a brief refresh of some definitions and concepts about data science.

- **Structured data** requires two key concepts:
  - **Data model** is a collection of concepts for describing data.
  - **Schema** is a description of a particular collection of data, using a given data model.
- The **Relational model** is on of the most common approach to manage data (SQL like) and can handle most of the data. A counter example is the facebook-like data which requires **graph model**. This model is made of 2 parts:
  - The **Schema**.
    For example, `Students(sid: string, name:string, age:integer)`
  - The **Instance**, *i.e.* the data at a given time.
    Definitons:
    - **Cardinality** is the number of rows. (Number of items)
    - **Degree** or **Arity** is the number of fields. (Number of attributes)
- Definitions of some "Database" terms:
  - A **JOIN** is a mean to combine tables based on shared attributes (most of the time some **IDs**). Despite its apparent simplicity beware of the many ways to compute a JOIN and check what is the default JOIN of a language before using it. The FIG 3 summarises these possibilities.
  - **Aggregation**, **reduction**, and **groupby** are the action of reducing data with a common opperation (**sum**, **count**, **average**, ...) to summarise them.
- Definitions of some "Pandas" terms:
  - **Series** are a name, ordered dictionary
    - keys are indexes
    - built on **numpy.ndarray** (so values can be any Numpy data type)
  - **DataFrame** is a table with named column
    - the columns are series
    - it is indeed a dictionary with (columnName $\rightarrow$ series)

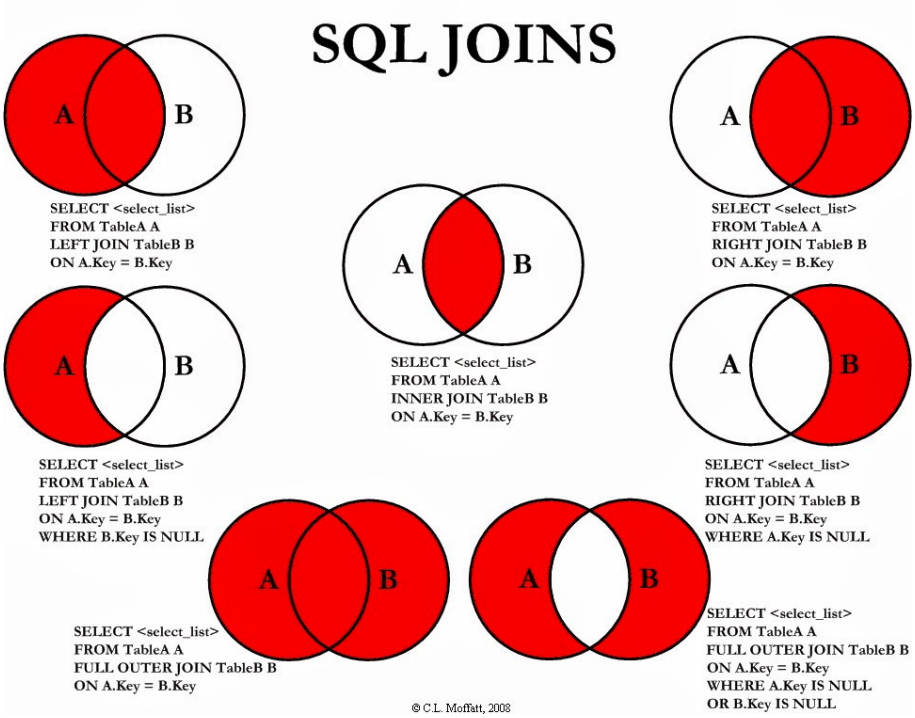


Figure 3: Different ways to join two tables and the related SQL command.

2.1 Panda vs SQL

Panda is built to allow easy and fast **data exploration** and not to be a database manager, as SQL is. Thus there are benefits and drawbacks of using it.

Pros	**Cons**
Lightweight & fast	Tables stored directly in memory
Great expressivness (combine SQL + Python)	No post-load indexing functionality
Easy plot for data visualization (eg Matplotlib)	No transactions, journalings
	Large, complex joins are slower

2.2 OnLine Analytical Processing (OLAP cubes)

OLAP tools enable users to analyze multidimensional data interactively from multiple perspectives. Conceptually, it is like an n-dimensional spreadsheet (a cube) on which we can apply various opperations to take decisions.

OLAP cubes are an other way to see data table and are contructed based on them, as shows FIG 4.

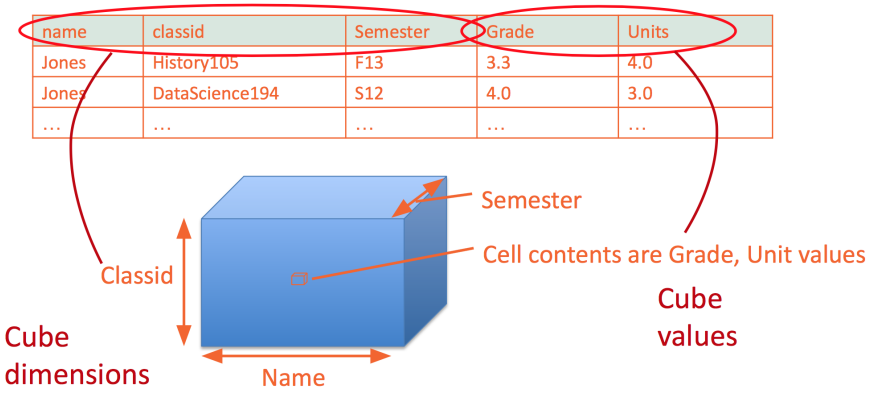
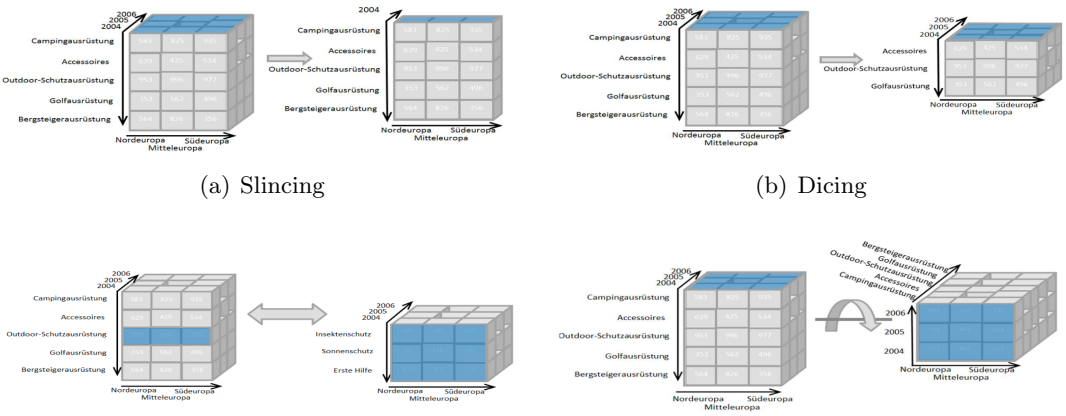


Figure 4: Construction of an OLAP cube from a table.

Operations on OLAP cubes are the following and are illustrated on FIG 5

- **Slicing** fixes one or more variable
- **Dicing** selects a range of one or more variable
- **Driling up/down** changes levels of a hierarchically-indexed variable, ie "zoom" on a variable and see the sub-categories it contains.
- **Pivoting** change the point of view of the cube. Swap an aggregated variable an a detailed one.



(a) Slincing (b) Dicing

(c) Driling up/down (d) Pivoting

Figure 5: Operations on OLAP cubes

Pros	Cons
The main adventage of OLAP cubes is that their are **conceptualy simpler** to understand by a non-scientist person, eg a business man who have to take day-to-day decisions based on company's data. Aggregations are limited but cover the main common cases that we can encounter.	Because of the "on-line" behaviour of this approach, all type of aggregation must be pre-calculated amoung all combination of axis which is very **expensive in memory and in time** (when updating the data)

3 Data Wrangling

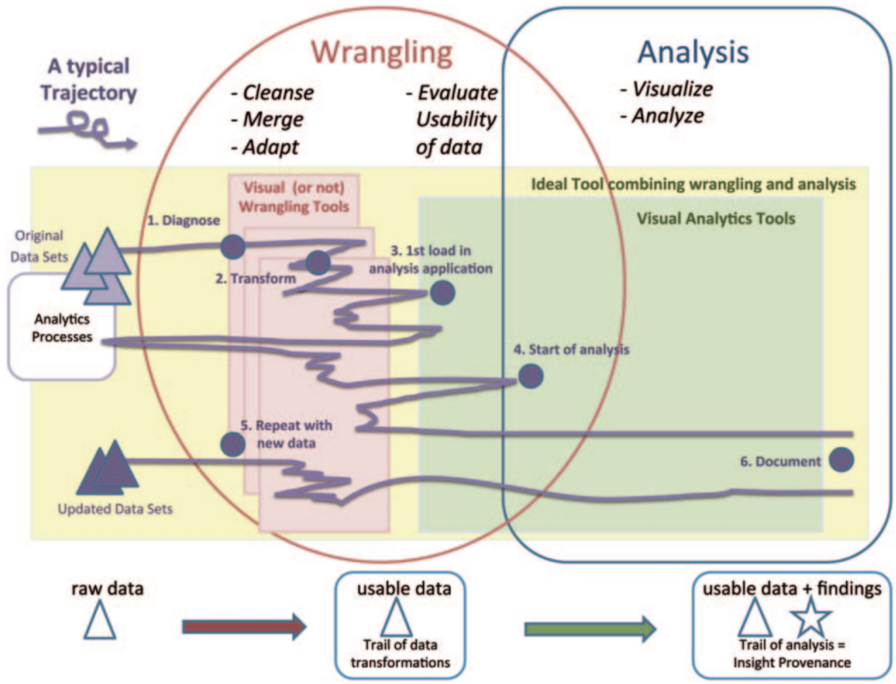


Figure 6: Things do not always happen as expected...

Before any analysis, data need to be transformed from "dirty" to clean and processable data.
Data comes from different sources (excel or SQL?), sometime collected through different methods over time, with different conventions (space or NaN?), etc ... Data wrangling's goal is to **extract and standardize these raw data**. The best way to do it is to **combine automation with visualizations** in order to find outliers.
Data's problem can come from (non-exhaustive):

- Missing data
- Incorrect data
- Inconsistent representations of the same data

- Non-standardized data (centimeter or inches? farenheit or celsuis ?)
- Duplicated data

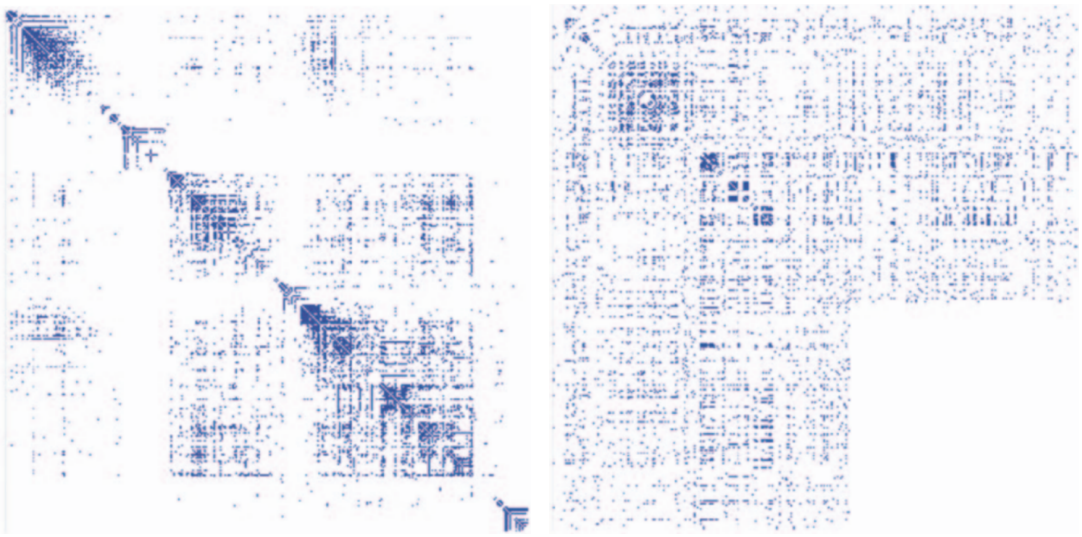
About 75% of theses problem will need **human intervention** to be corrected (by the data-scientist or by crowdsourcing).
Even if it seems really dirty, **beware not to over-sanitize the data!**. Applying what we can call "defensive programming" is not a good idea because we risk to lose any interesting data, keeping only the ones that fit perfectly in our model.

3.1 Diagnosis of the data

One of the most important aspect of Data Wrangling is to **understand** the data and to **find possible problems**. In order to "diagnose" the data, two tools can be used:

- **Visualization** (A *toughtful* visualization will always help)
- **Basic Statistics**

Matrix visualizations of the facebook graph is shown in Figure 7. The Relational visualization, Figure 7(a), does not show any particular problem in the data. But the Time dependant visualization, Figure 7(b), shows that the Facebook API reached its limit while collecting data.



(a) Relational visualization (b) Time dependant visualization

Figure 7: Matrix visualization of the facebook graph.

3.2 Dealing with missing values

Values can often miss from the data we have, because of various events (war, fire, ...). We must detect and correct these values with different method according with the domain we are working in.

Whatever the method used, it's good to keep track of these changes to know which are original data and which are modified ones.

- Set values to zero FIG 8(a)
- Interpolate based on existing data FIG 8(b)
- Omit missing data FIG 8(c)
- Interpolation with track kept 8(d)

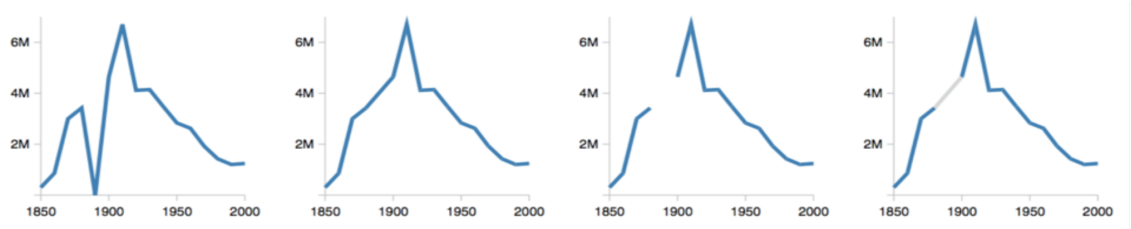


Figure 8: To deal with missing values.

3.3 General procedure

Once the data are well wrangled and before trying to analyse them we must take care of two more steps:

1. **Deal with uncertain data** (can arise from measurement errors, wrong sampling strategies, etc.)
2. **Parse/trasform data** (with aggregation and reduction techniques) to obtain meaningful records

It's always ideal to have the code and/or the documentation about the dataset you are analyzing (provenance).

4 Data Variety

The "3 Vs" of Big Data: *Volume*, *Velocity* and *Variety*. In this course, we don't address the *Volume* and *Velocity* parts (A course on Database does). Since there is a lot of variety in the data, we need to prepare the data. This flow is called **ETL**:

- **Extract** from the *source(s)*.
- **Transform** data at the source, sink, or in a *staging area*.
- **Load** data into the *sink*.

This variety of the data comes, in a first place, from the many different sources from which we extract them: *files, databases, logs, ...* Each of these sources uses (or not!) its proper convention and can contains structured (DB), semi-structured (logs) or unstructured (web page) data.

4.1 Role of Schema

The **Schema**, which specifies the *structure* and *types* of data repository, is changing. Traditional databases are **schema-on-write**, *i.e.* you cannot load data into a table without a schema. But new data stores (NoSQL for example) are **schema-on-read** or **schemaless**.

- **Schema-on-write** is typically SQL, where we must create a table before inserting data in our system. Data must scale the defined schema and this is both the strength and weakness of the system. Strength because the data is perfectly oriented and respect the constraints we establish. Weakness because schemas are always subjective in some ways and data (which are perfectly corrects) may not fit with it.

- **Schema-on-read** is for instance XML, where you create the schema according with the data you read.

- Youtube and Google Cache where the first **schemaless** data system. Without schema, everything is simply stored as a string and we need a parser to return a typed data.

4.2 Examples of data

4.2.1 XML and DOM

The XML data are used mostly with HTML and specifies the data structure. An XML schema can be applied to interpret the XML data and specifies the **data types**. Figure 9 shows the XML data 9(a) and the schema 9(b) used to parse and type the data.

```
<location>
  <latitude>37.78333</latitude>
  <longitude>122.4167</longitude>
</location>
```

(a) XML data

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="unqualified">
<xsd:complexType name="location">
  <xsd:sequence>
    <xsd:element name="latitude" type="xsd:decimal"/>
    <xsd:element name="longitude" type="xsd:decimal"/>
  </xsd:sequence>
</xsd:complexType name="location">
```

(b) XML schema

Figure 9: Example of XML.

The XML is a text format that encodes **DOM** (Document-Object Models). It's a data structure often used by Web pages. The DOM is tree-structured. An example of a DOM is given in Figure 10.

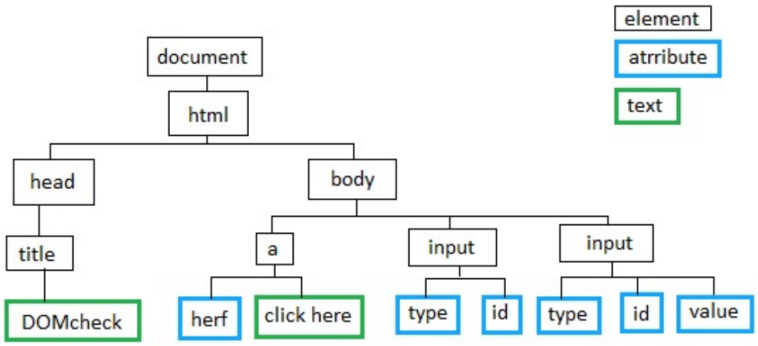


Figure 10: Example of a DOM tree for an HTML Web page.

THe XML schema allows a database to interpret the data when running queries. It can do arithmetic or range queries on numerical values, for example.

4.2.2 JSON

JSON stands for Javascript Object Notation. It's a schemaless data (schema support was added later). An example of JSON data is shown in Figure 11

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
       "streetAddress": "21 2nd Street",
       "city": "New York",
       "state": "NY",
       "postalCode": "10021-3100" },
  "phoneNumbers": [
      { "type": "home",
        "number": "212 555-1234" },
      { "type": "office",
        "number": "646 555-4567" } ],
  "children": [],
  "spouse": null
}
```

Figure 11: Example of a JSON data.

JSON is typically used to represent **hierachical data structures** directly in the target language (Javascript or Java at the beginning). The transformation on the data are procedural in the target languages. It is often easier for some tasks, but it can be painful for some of them: for example schema changes.

4.2.3 Tabular data

A Tabular Data is simply data put into a table such as CSV or TSV. Definition of a table:

- A **table** is a collection of **rows** and **columns**.

- Each row has an **index**.
- Each column has a **name**.
- A **cell** is specified by an (index, name) pair.
- A cell may or may not have a **value**.

It's a very simple yet powerful data type. For example, the sensors usually output data in the form of time series, transformed into a tabular format. However, a system dealing with sensor data should:

- support both long-term (**trend**) and short-term (**real-time**) queries
- have **low latency** but also efficient. It should use **real-time indexing** for longer-term queries.
- support triggers (**alerts**) for a variety of conditions.

Therefore, the **complexity of a data format** does not determine the **complexity of the system required to properly handle it**.

4.2.4 Log files

The log files are simple text files giving information about process. The daemons, such as `httpd`, `mysqld` or `syslogd`, usually create logs. `syslog` was developed by Eric Allman. It's a way for devices to send event messages to a server that will log all the events. Splunk is a company that built a successfull business model around the syslog events.

4.2.5 Binary formats

They are often the key to performance because we **avoid expensive parsing**. The modern formats even support nested structures, various level of schema enforcement, **compression**, etc. Somes examples: Protocol Buffers (Google), Avro, Parquet, etc.

4.3 Processing the data (JSON and XML)

In order to process XML, we can use the DOM. It can also be used to process JSON data. The DOM is very easy to work with: all the data are directly accessible by links. The problem is that we **might not care about most of the data** and if the data are big, they **might not fit into the RAM**. In order to deal with these two problems, we can use a **SAX** parser which is an event-driven parser. It will find all the **open-close-tag events** in an XML document and will bf do callbacks to user code.

+ User code can respond to only a subset of events corresponding to the tag it is interested in.
+ User code can correctly compute aggregates from the data rather than create a record for each tag.

+ User code can implement flexible error recovery strategies for ill-formed XML.

– User code must implement a state machine to keep track of "where it is" in the DOM tree.

For JSON, most parsers construct the "DOM" directly. But there are a few SAX-style parsers: Jackson, JSON-simple, etc. Sometimes **SAX-style is the way to handle ill-formed datasets**, an endless array of objects for example.

4.4 HTML and Web Services

4.4.1 HTML

Internet contains an "enormous" amount of data. Some crawlers such as Common Crawl dataset contains about 1.82 billions web pages (for 145 TB). We can use different tools to crawl data from the web. Examples for Python: Beautiful Soup, Requests, Scrapy, etc.

Most of the time, the Web pages are considered as unstructered data. But you can find some semi-structured data, *e.g.* Google WebTables. Some big "internet" companies (Google, Yahoo, Yandex and Microsoft) are sponsoring a project called **schema.org** to create structured or semi-structured Web pages. A core vocabulary for the type of fields is given. schema.org is more and more used. It's also used by knowledge bases such as Google Knowledge Graph. **WikiData** is a community project to create an open database of structured data taken from Wikipedia.

4.4.2 Web Services

Screen-scraping the content of a large website was possible before, but become more and more difficult nowadays. This is mainly due to the content "hidden" behind a form or an authentification. Take for example facebook without account, or the IS-academia course page if you do not select a semester. Therefore big companies are providing Web Service APIs[1] to retrieve data from their website. There are two kinds of Web Services:

- The old way: XML-based RPC-style messages: SOAP
- The new way: REST-style stateless interactions, URLs encode state

4.4.2.1 RPC

The SOAP RPC [2] messages typically encode arguments that are presented to the calling profram as parameters and return values. HTTP POST/GET are used to communicate.

[1] Application Program Interface: Set of subroutine definitions, protocols, and tools for building software and applications. In this particular case, the APIs are used to retrieve the data from the Web page, *e.g.* Facebook API to retrieve the contacts.

[2] SOAP = Simple Object Access Protocal, RPC = Remote Procedure Call

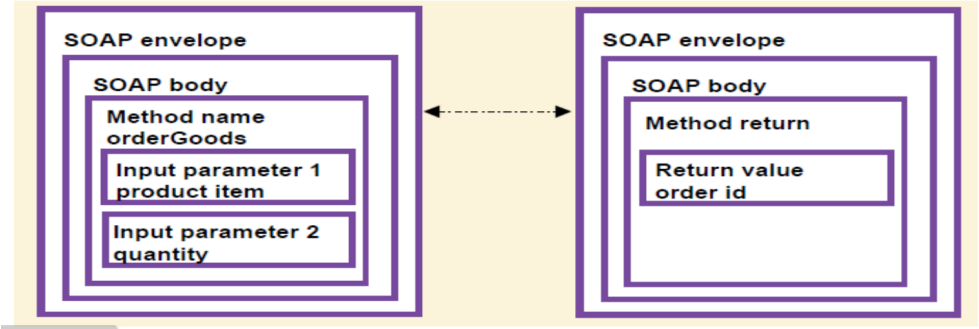


Figure 12: Example of a SOAP RPC exchange.

This kind of procedure (same for XML-RPC) requires a request-response cycle. This often leads to longer "conversations". The RPC-style is being quickly superseded by newer and more user-friendly technologies.

In **RPC systems**, the design emphasis is on **verbs**. It uses functions such as *getUser()*, *addUser()*, etc.

4.4.2.2 REST

REST[3] is a **stateless** client/server protocol. The principles are:

1. Each message in the protocol contains all the information needed by the receiver to understand and/or process it. This constraint attempts to *"keep things simple"* and avoids needless complexity.
2. Set of Uniquely Addressable Resources
   - *"Everything is a Resource"* in a RESTful system
   - Requires universal syntax for resource identification, *e.g.* URI.
3. Set of Well-Defined Operations that can be applied to all resources
   - In the context of HTTP (REST APIs), the primary methods are:
     POST, **GET**, **PUT**, and **DELETE**
     These are similar (but not exactly) to the database notion of CRUD (Create, Read, Update, and Delete)
4. The use of Hypermedia both for Application Information and State Transitions
   - Resources are typically stored in a structured data format that supports hypermedia links, such as XHTML or JSON.

In **REST systems**, the design emphasis is on **nouns**. It uses the HTTP Protocols (POST, GET, PUT, and DELETE) a *User*, a *Location*, etc.

[3] REpresentation State Transfer