Data Processing

PRI 24/25 · Information Processing and Retrieval
M.EIC · Master in Informatics Engineering and Computation

Sérgio Nunes
Dept. Informatics Engineering
FEUP · U.Porto

Today's Plan

- Focus on data processing
  - Practical perspective
  - Storage solutions
  - UNIX philosophy
  - Makefiles for data pipelines
- Data presentation

- Review "Milestone 1 - Data Preparation"
- Next lab class

Data Processing

Data Storage

Data Models

- Data models have a role part in software development, not only how software is implemented but also on how we think about the problem.
- Defining abstraction layers is a common pattern in software architectures, where each layer hides the complexity of the layers below.
- When thinking in data models, each layer hides the data complexity of previous layers, and provides a cleaner data model.
- Different data models make different operations faster/slower, natural/awkward, possible/impossible.
- Models discussed: relational model, document model, and graph-based models.

Relational Model

- The relational model is the best-known and most dominant data model.
- In the relational model data is organized in *relations* (called *tables* in SQL), where each relation is an unordered collection of *tuples* (*rows* in SQL).
- The NoSQL (*Not Only SQL*) movement emerged in the 2010s as a need to provide greater scalability, specialized query operations, and a more flexible and expressive data model.
- The "*object-relational mismatch*" (or *impedance mismatch*) is an expression used to represent a common criticism to the SQL data model — the need of a translation layer between "tables, rows, and columns" and the "objects" used in object-oriented programming.

Document Model

- In the document models data is organized as single instances (documents).
- In document databases, nested records are stored together with their parent records.
- Also known as document stores, document-oriented databases.
- Advantages of document data models are:
  - schema flexibility;
  - better performance due to locality (i.e. related data is stored together);
  - for some applications, the data model is closer to the application's data structures.
- The relational model is better suited to support joins, N-1, and N-N relationships.

Relational vs Document Representations

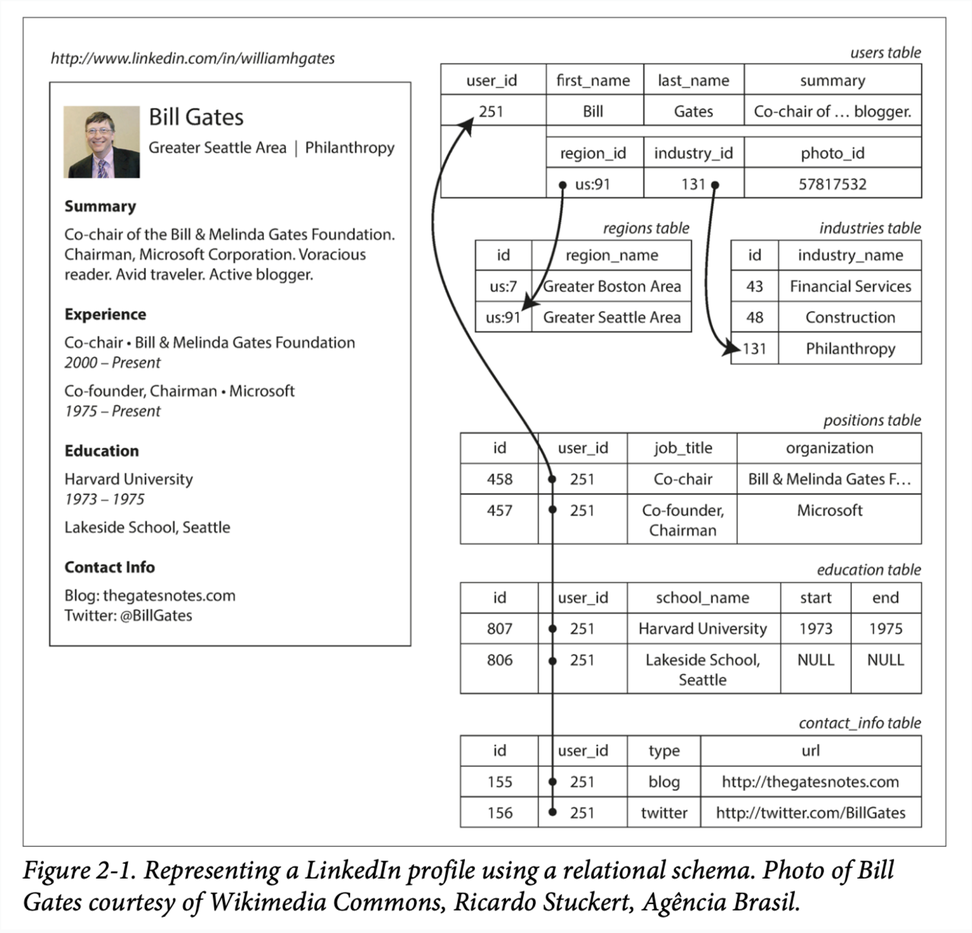


Figure 2-1. Representing a LinkedIn profile using a relational schema. Photo of Bill Gates courtesy of Wikimedia Commons, Ricardo Stuckert, Agência Brasil.

Example 2-1. Representing a LinkedIn profile as a JSON document

```
{
  "user_id":     251,
  "first_name":  "Bill",
  "last_name":   "Gates",
  "summary":     "Co-chair of the Bill & Melinda Gates... Active blogger.",
  "region_id":   "us:91",
  "industry_id": 131,
  "photo_url":   "/p/7/000/253/05b/308dd6e.jpg",

  "positions": [
    {"job_title": "Co-chair", "organization": "Bill & Melinda Gates Foundation"},
    {"job_title": "Co-founder, Chairman", "organization": "Microsoft"}
  ],
  "education": [
    {"school_name": "Harvard University",       "start": 1973, "end": 1975},
    {"school_name": "Lakeside School, Seattle", "start": null, "end": null}
  ],
  "contact_info": {
    "blog":    "http://thegatesnotes.com",
    "twitter": "http://twitter.com/BillGates"
  }
}
```

Materials from Kleppmann, M. Designing Data-Intensive Applications (2017).

Schema Flexibility

- Most document databases do not enforce any scheme on the data in documents.
- No schema (*schemaless*) means that arbitrary keys and values van be added to a document and, when reading, clients have no guarantees as to what fields to expect.
- Although schemaless databases do not impose a schema, code that reads the data usually assumes some kind of structure, thus there is an implicit schema. But not enforced by the database!
- This solution is advantageous when the items in a collection don't have the same structure, or the structure of data is determined by external systems which may change over time.
- Where the schema is enforced is the key issue — schema-on-read (the structure is enforced when data is read, at code level); schema-on-write (the structure is enforced by the database, the traditional relational approach).

Property Graph Model

- When the application has mostly one-to-many relationships (i.e. tree-like structures) or no relationships between records, the document model is appropriate.
- If many-to-many relationships are very common, and the connections within the data are complex, it is more natural to model data as a graph.
- A graph has to kinds of objects: vertices (also known as nodes or entities) and edges (also known as relationships or arcs).
- Graphs are not limited to homogeneous data, an advantage of the graph model is the fact that it provides a consistent way of storing completely different types of data in a single datastore (e.g. people, places, events).
- Typical examples, include: social graphs, web graph, transportation networks.

Graph Example

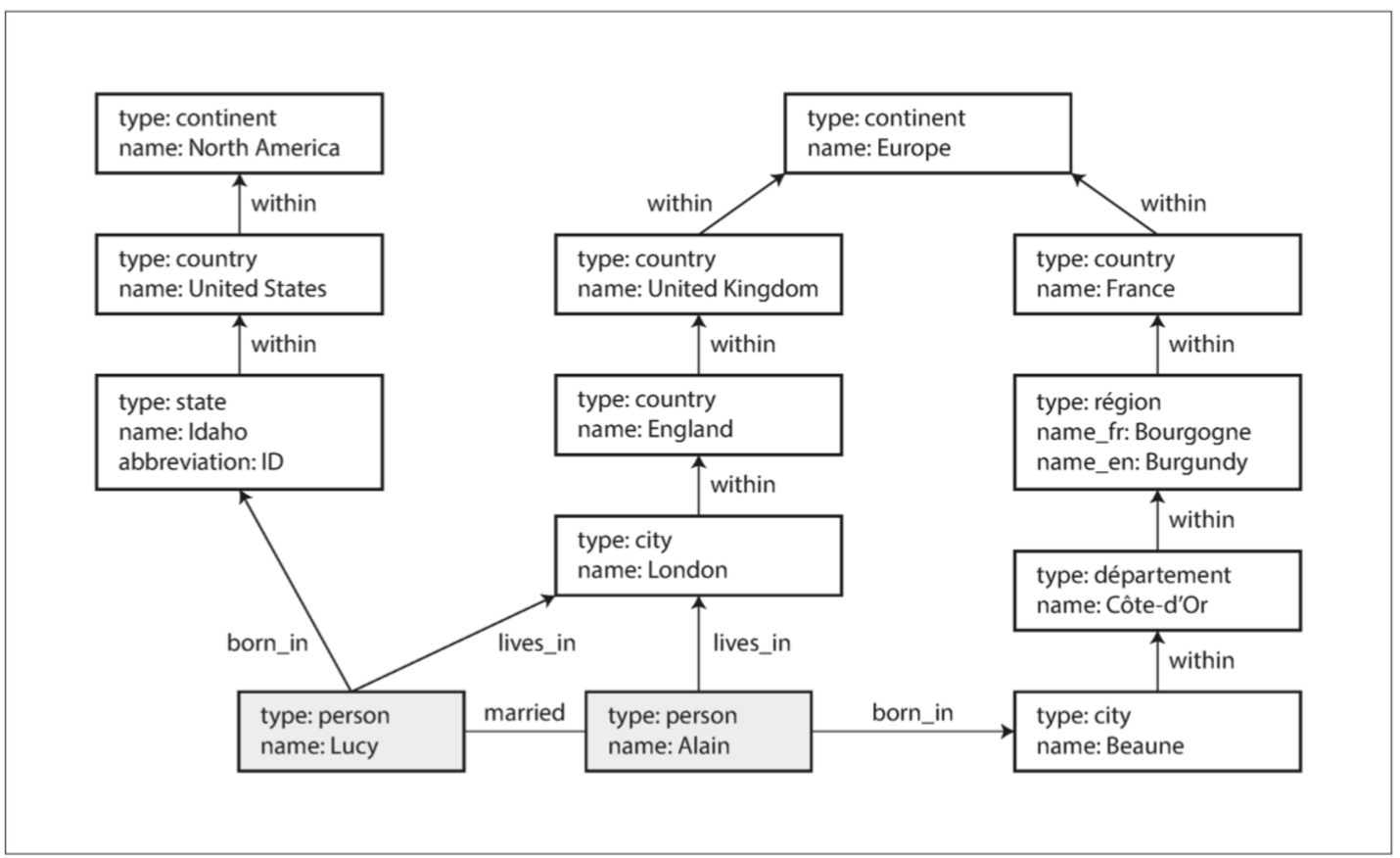


Figure 2-5. Example of graph-structured data (boxes represent vertices, arrows represent edges).

Image from Kleppmann, M. Designing Data-Intensive Applications (2017).

Triple-Store Model

- Similar to the property graph model.
- Important because there are various relevant tools and languages for triple-stores.
- In a triple-store, all information is stored in the form of three-part statements: *(subject, predicate, object)*. For example: *(Maria, enrolled, PRI)*; *(PRI, course, M.EIC)*.
- The subject of a triple is equivalent to a vertex in a graph.
- The object is either, a value (primitive datatype) or another vertex in the graph.
- RDF is a data model for specifying data subject-predicate-object (triples) statements.
- SPARQL is the querying language for triple-stores using the RDF data model.

Data Storage Solutions

- Industry-grade open-source solutions exist for the main paradigms.
- Relational model
  - SQLite, www.sqlite.org
  - PostgreSQL, www.postgresql.org
  - MariaDB, mariadb.org
- Document model
  - MongoDB, www.mongodb.com
  - CouchDB, couchdb.apache.org
- Graph model
  - Neo4j, neo4j.com

Data Storage Summary

- Data models are central in data processing.
- The relational data model is the most widely used approach.
- Non-relational solutions have emerged in recent years under the "NoSQL" umbrella.
- Two main alternative directions — document databases and graph databases.
  - With document databases, data is mapped to self-contained documents and relationships between documents are rare.
  - With graph databases, data is separated in independent units and, potentially, everything is related to everything.
- These three models are widely used and each is good in its respective domain.
- There is no one-size-fits-all solution, that's why different systems exist. Also, database systems support features from "competing" models, e.g. JSON support in relational database systems.

Batch Processing

Interaction Types

- There are three main approaches in building the interaction style in data processing.
- **Online** systems (services), where a service waits for requests or instructions from a client to arrive. When a request is received, the service tries to handle as quickly as possible and send the response back. Response time is the primary measure of performance.
- **Offline** systems (batch processing), where a batch processing system takes a large amount of input data, runs a job to process it, and produces some output data. Suitable for long jobs or asynchronous processes.
- **Stream** processing systems, where a service operates on inputs and produces outputs (rather than responses) as a result of an event happening (rather than periodically at scheduled times).

Batch Processing

- Simple log analysis with Unix tools.
  - ```
    cat /var/log/nginx/access.log | # read the log file
    awk '{print $7}' | # split each line into fields and take the 7th (URL)
    sort | # alphabetically sort each URL
    uniq -c | # filter duplicate URLs and count the number of duplicates
    sort -r -n | # numerically sort based on the number of occurrences
    head -n 5 # print the top 5 lines
    ```
- Powerful and versatile solution that can quickly process gigabytes of log files.
- An alternative approach would be to build a custom program to read each line into a hash table and keep track of each URL occurrence.
- The main difference between the two approaches is the fact that the custom program loads all data into memory, while the Unix approach relies on sorting as a key step in preparing data for counting.
    ```

The Unix Philosophy

- The *Unix Philosophy* encapsulates a set of design principles that became popular among the developers and users of Unix (1960/70s).
  - 1. Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new features.
  - 2. Expect the output of every program to become the input to another, as yet unknown, program. Don’t clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don’t insist on interactive input.
  - 3. Design and build software, even operating systems, to be tried early, ideally within weeks. Don’t hesitate to throw away the clumsy parts and rebuild them.
  - 4. Use tools in preference to unskilled help to lighten a programming task, even if you have to detour to build the tools and expect to throw some of them out after you’ve finished using them.
- Many of this concepts are relevant and important when designing systems today.

Unix Uniform Interface

- In Unix, everything is a file, i.e. an ordered sequence of bytes.
- Actual filesystem files, TCP connections, device driver, etc., all share the same uniform interface, and thus can be plugged together.
- This makes it easy to build systems that "pipe" together many different blocks.
- Additionally, in Unix there is a decoupling of logic and wiring, in the send that the user can wire up the input and output in whatever way they want. This make it easier to compose small tools into bigger systems.
- One important limitation is that Unix tools run only on a single machine — tools like Hadoop have appeared to overcome this limitation.

Makefiles

Managing Data Pipelines

- A data processing project is typically a fragmented process, involving multiple operations, data storages, files, folders, formats, etc.
- Managing these processes and workflows is a requirement. The goal is to produce "software artifacts" that are maintainable, versionable, testable, and collaborative.
- Solutions include Apache Airflow, Make, Nextflow.
  - https://airflow.apache.org/
  - https://www.gnu.org/software/make/
  - https://www.nextflow.io/

Make

- Make is a software tool that controls the execution of commands to derive *target files*, based on the existence (or change) of other files (*dependencies*) and the execution of operations (*recipes*).
- Make was originally built, and is still a central piece, in software building processes.
- It can generally be used in scenarios where the execution workflow can be encapsulated as a set of dependencies and execution rules.
- Make is able to automatically decide what to execute based on direct or indirect dependencies to files that have changed.
- Make is language agnostic and follows a declarative programming paradigm.
- Use cases: software building, scientific publishing (figures, tables, LaTeX files, etc), data pipelines.

Makefiles

- Makefile is a text file where Make execution rules and dependencies are defined.
- The default filename is "Makefile", with no extension and starting with an uppercase.
- A makefile consists of a series of rules.
  - Each rule defines a target name (or list) and an optional list of prerequisites.
  - Additionally, a rule can have a list of commands to execute (recipe).
  - Recipe commands must be indented with TABs.
- To build a specific target, "make <target>" is used.
- Manual online: https://www.gnu.org/software/make/manual/make.html

Example Makefile

```
# Makefile

# Vars
URL = "https://en.wikipedia.org/wiki/Alice%27s_Adventures_in_Wonderland"

# Rule 'all' defines how to build everything
all: alice.txt

# Rule to obtain "alice.html"
# The existence of specific tools can be checked on the dependencies (e.g. curl)
alice.html: /usr/bin/curl
  curl -s $(URL) > alice.html

# Rule to generate "alice.txt" from "alice.html"
alice.txt: alice.html
  lynx -dump alice.html > alice.txt

# Rule to remove generated files
clean:
  rm -f alice.txt alice.html
```

```
$ make alice.html # To execute a specific rule

$ make # To build all. If alice.txt already exists, it won't build it again
$ make -s # Silent execution

$ make clean # To clean up
```

Makefiles Highlights

- Makefiles support control structures such as if conditions, execution cycles, etc.
  - https://www.gnu.org/software/make/manual/html_node/Conditionals.html
  - https://www.gnu.org/software/make/manual/html_node/Foreach-Function.html
- Make can build targets in parallel. Use the -j flag to take advantage of multiple cores.
- Automatic variables provide access to values computed for the rule in execution:
  - Examples: $@ (first target); $^ (all prerequisites);
  - https://www.gnu.org/software/make/manual/html_node/Automatic-Variables.html
- It is possible to store the execution of a shell command in a variable
  - PDF_FILES = $(shell find . -type f -name '*.pdf')

Makefiles as Macros

- You can use targets to simply encapsulate multiple commands, e.g.
  - 'make requirements' to setup your environment
  - 'make publish' to upload to a remote server
  - 'make web' to start a local web server and server your project
  - 'make figures' to prepare figures from your data collection
  - ...
- Rules that should always be executed are marked as a 'phony target', with:
  - .PHONY requirements publish

Makefiles - Additional References

- **Makefile Tutorial**, 2021
  https://makefiletutorial.com/
- **Chapter 6 Project Management with Make**, from Data Science at the Command Line, Janssens, J., O'Reilly, 2021
  https://www.datascienceatthecommandline.com/2e/chapter-6-project-management-with-make.html
- **Managing Project with GNU Make**, Robert Mecklenburg. O'Reilly Open Book, 2004
  https://www.oreilly.com/openbook/make3/book/index.csp
- **Why Use Make**, Mike Bostock, 2013
  https://bost.ocks.org/mike/make/
- **Reproducibility with Make**, from The Turing Way Community, Arnold, B. et al., 2019
  https://the-turing-way.netlify.app/reproducible-research/make.html
- **Chapter 36 Automating data-analysis pipelines**, from STAT 545 course, Jenny Bryan
  https://stat545.com/automating-pipeline.html

Data Presentation

Data Presentation

- Data presentation is an important task in data projects.
- Depending on the audience, presentation can differ greatly.
- Presentation can simply use tables to effectively communicate the key aspects.
- Or use more or less complex visualizations, static or interactive.
- We saw that data visualization can be an important tool in data preparation, e.g. to investigate data or evaluate data quality issues. In this case, the audience is mostly specialized.
- For other end-users, presentation solutions will depend, and can range from static images (e.g. in internal documentation), to publicly available websites (e.g. intranet), to interactive applications (e.g. data journalism).

Data Visualization

- Data visualization (*data viz*) refers to algorithmically drawn, easy to regenerate, and aesthetically simples visualizations. As opposed to infographics, which are manually drawn and aesthetically more complex visualizations.
- Data visualizations can be classified according to their complexity considering the number of dimensions they represent, i.e. the number of discrete data types visually encoded in the diagram.
- Visualization can also be divided in two categories, each with a different purpose: exploration and explanation.
  - Exploration is generally done at a higher level of granularity, earlier in the process, and the main goal is to understand what is in the data;
  - Explanation is appropriate when you already know the data and you want to support and present findings.
- In the context of PRI,
  - we are interest in simple (but insightful), static visualizations;
  - that help us understand and highlight relevant characteristics of the data;
  - and are programmatically obtained, i.e. use scripts to quickly generate the visualizations.

Napoleon's Russian Campaign (1869)

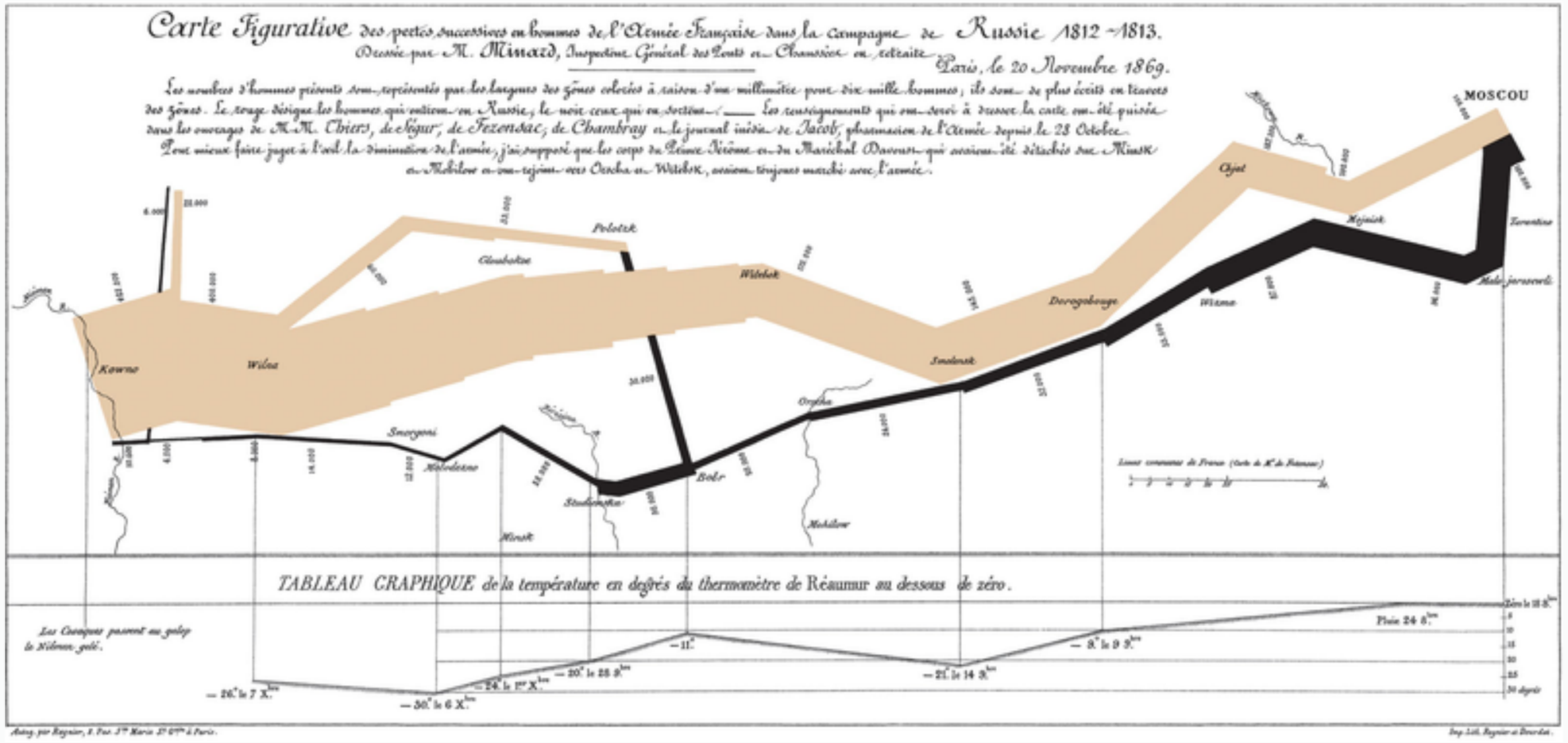


Image from Charles Joseph Minard, Wikipedia. https://en.wikipedia.org/wiki/Charles_Joseph_Minard

Cholera Map (1854)

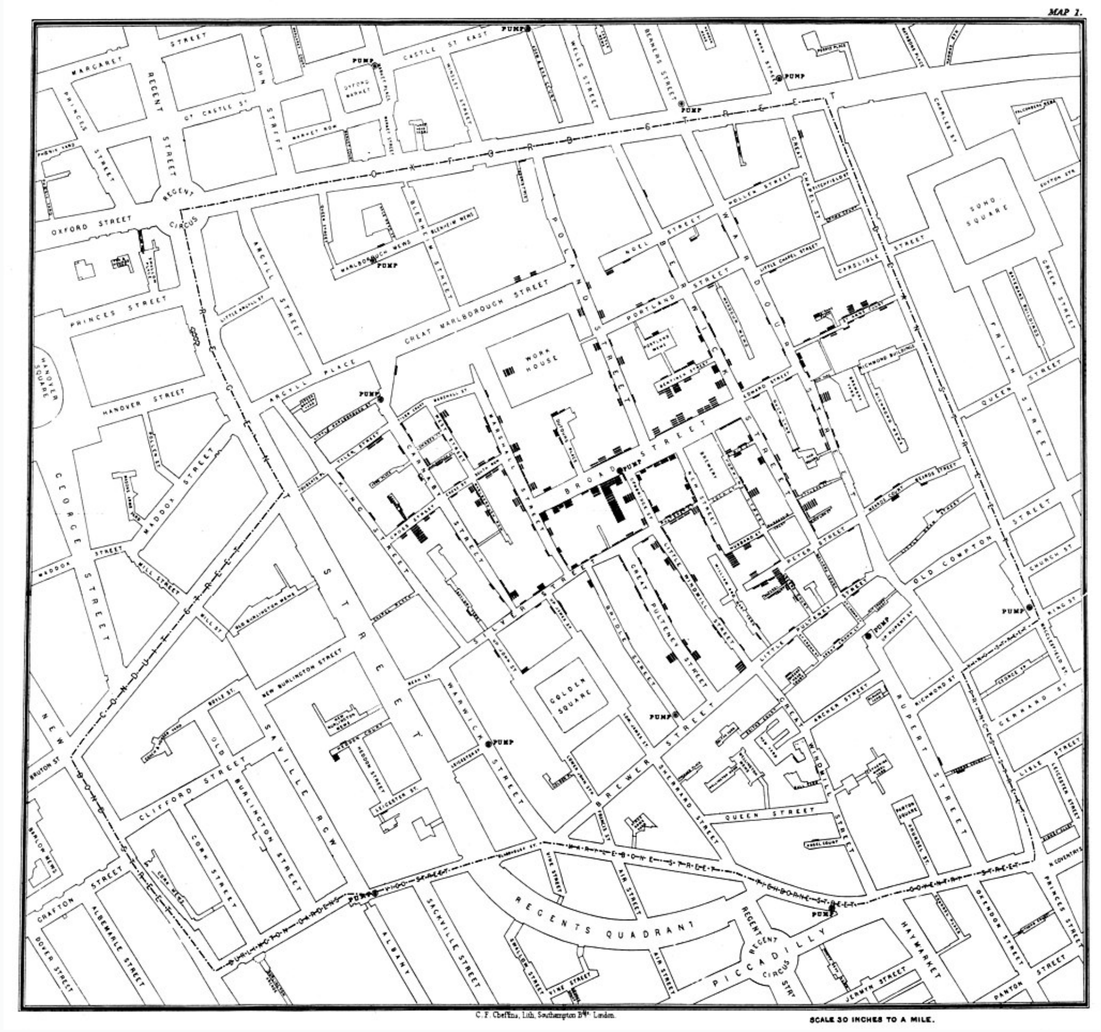


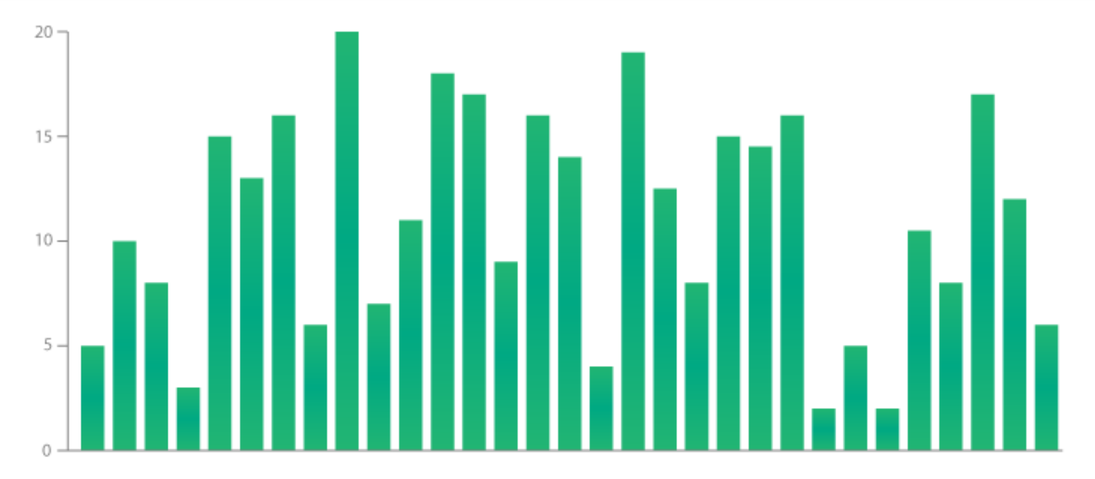
Image from John Snow, Wikipedia. https://en.wikipedia.org/wiki/John_Snow

Elements of Visualization

- Common data visualization solutions include:
  - **charts**, suitable for numeric data, especially when comparing data sources or groupings;
  - **time-series**, or timelines, help represent observations over time;
  - **maps**, focus on geographic properties of data;
  - **interactive**, where control is given to the user to explore different paths or options;
  - **words**, representing highlights (e.g. most frequent) of textual descriptions;

- The Data Visualization Catalogue, https://datavizcatalogue.com

Bar Chart



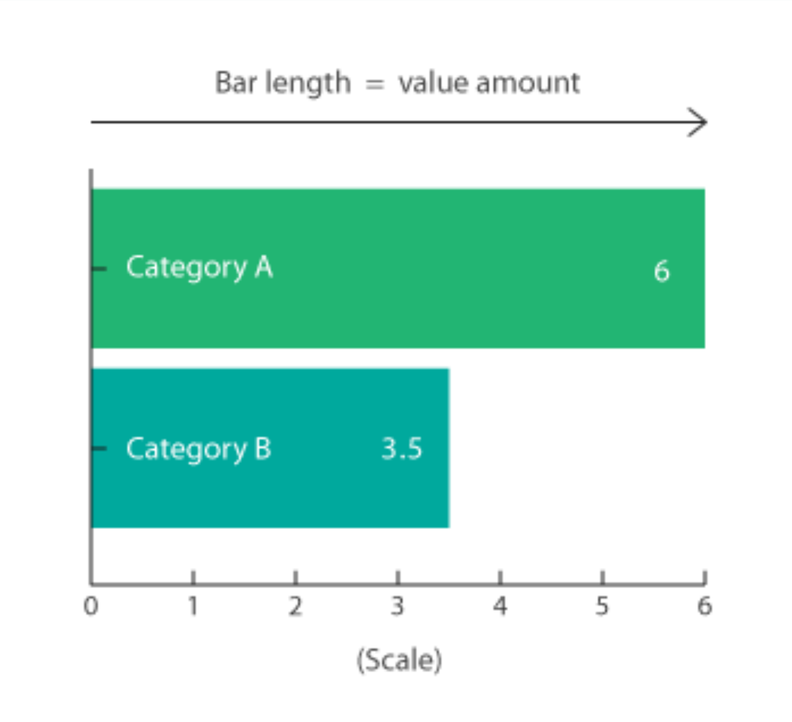
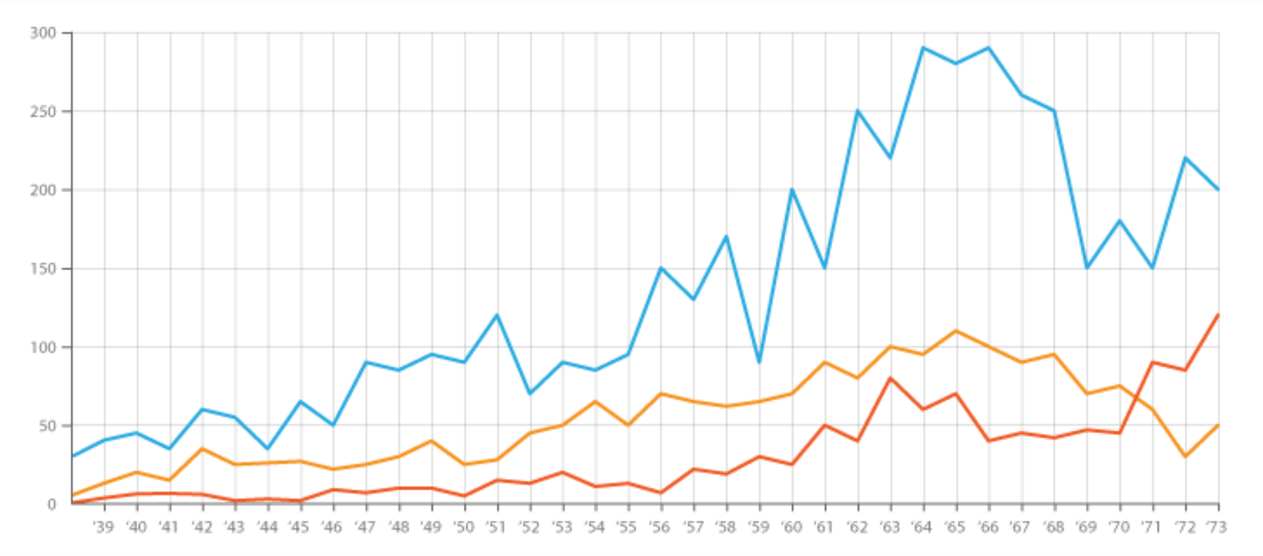


Image from Bar Chart, https://datavizcatalogue.com/methods/bar_chart.html

Line Graph



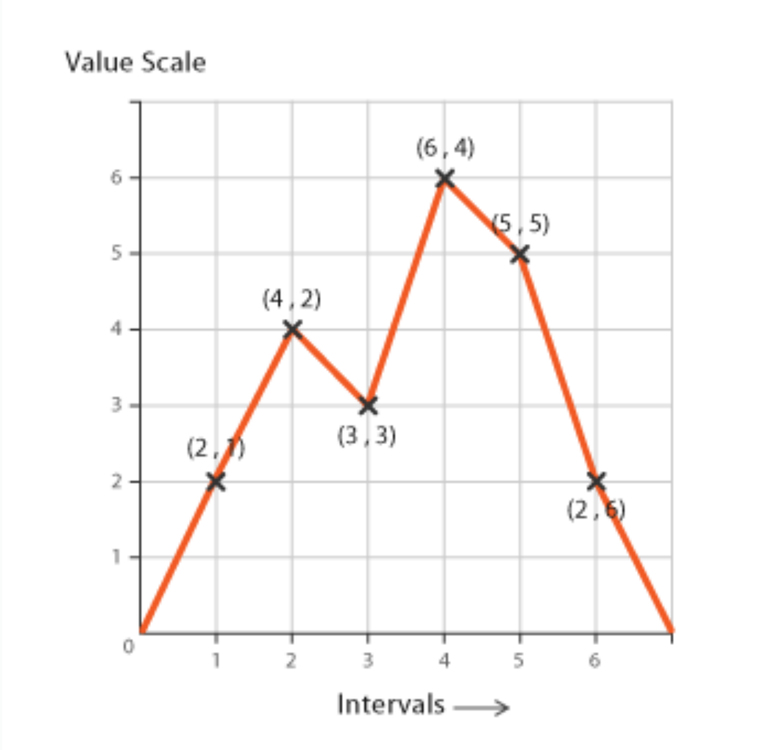
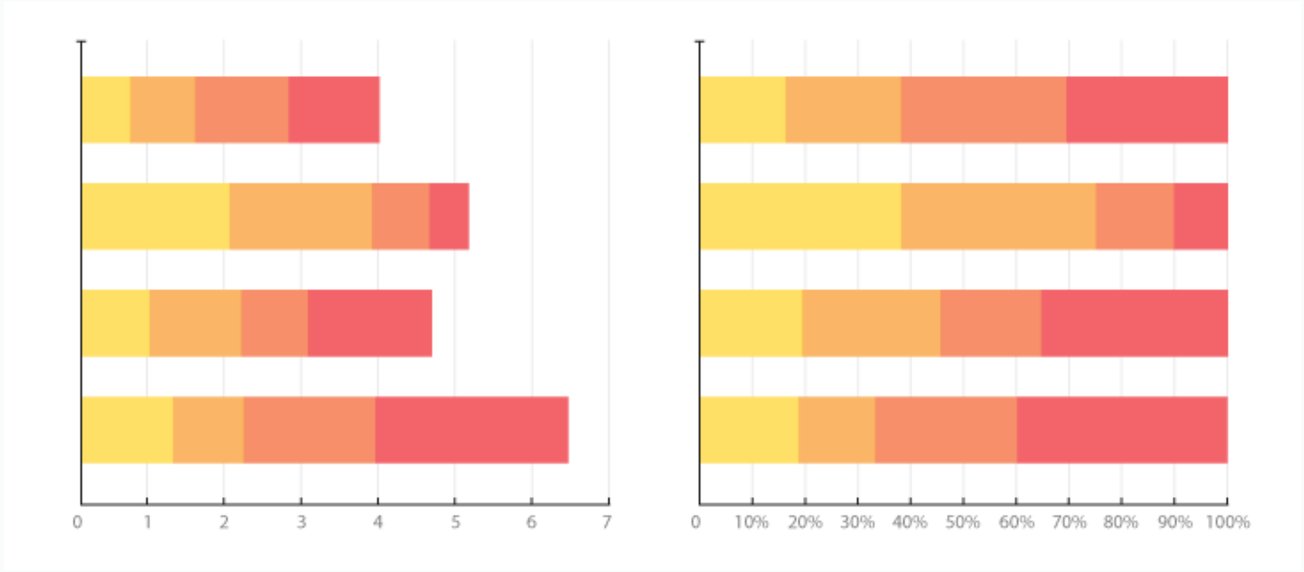


Image from Line Graph, https://datavizcatalogue.com/methods/line_graph.html

Stacked Bar Chart



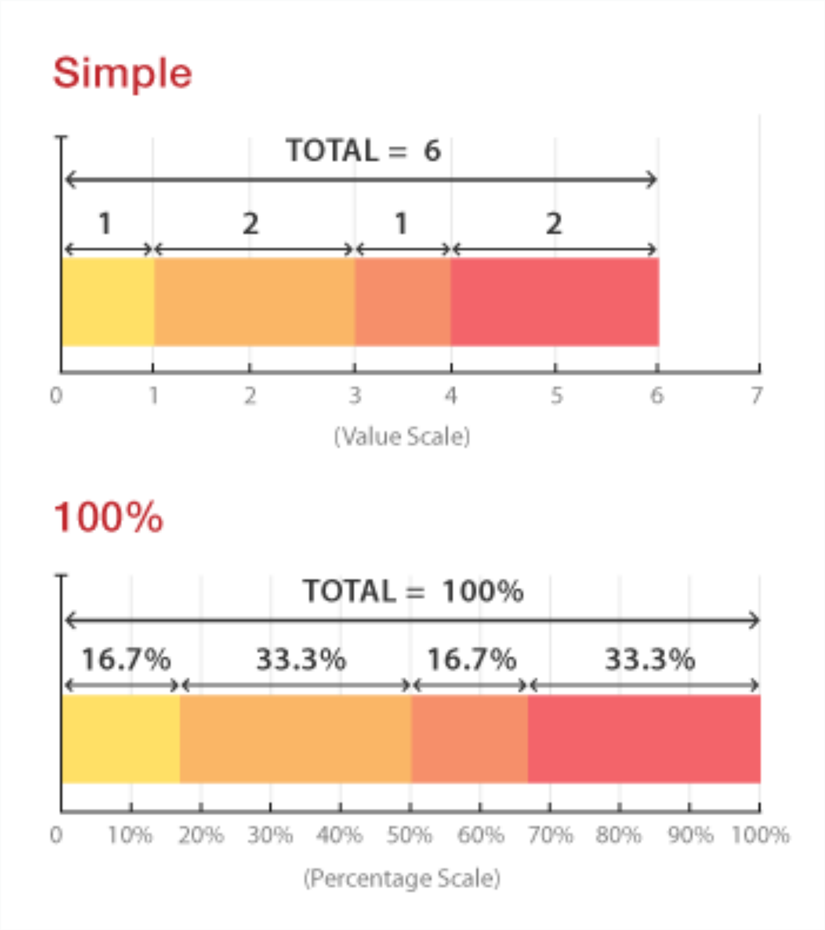
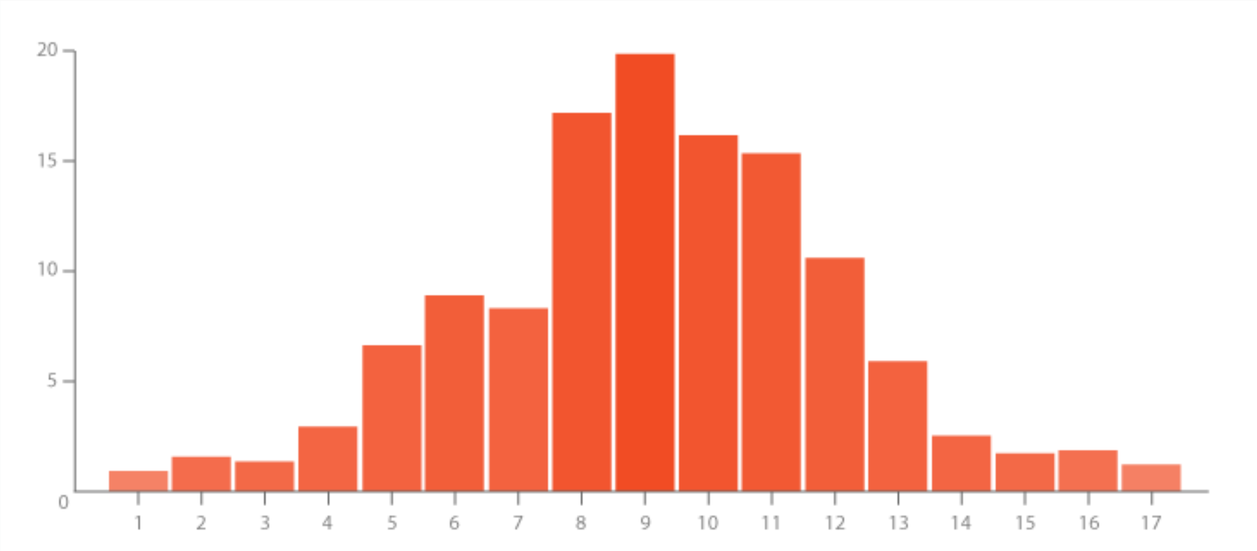


Image from Stacked Bar Chart, https://datavizcatalogue.com/methods/stacked_bar_graph.html

Histogram



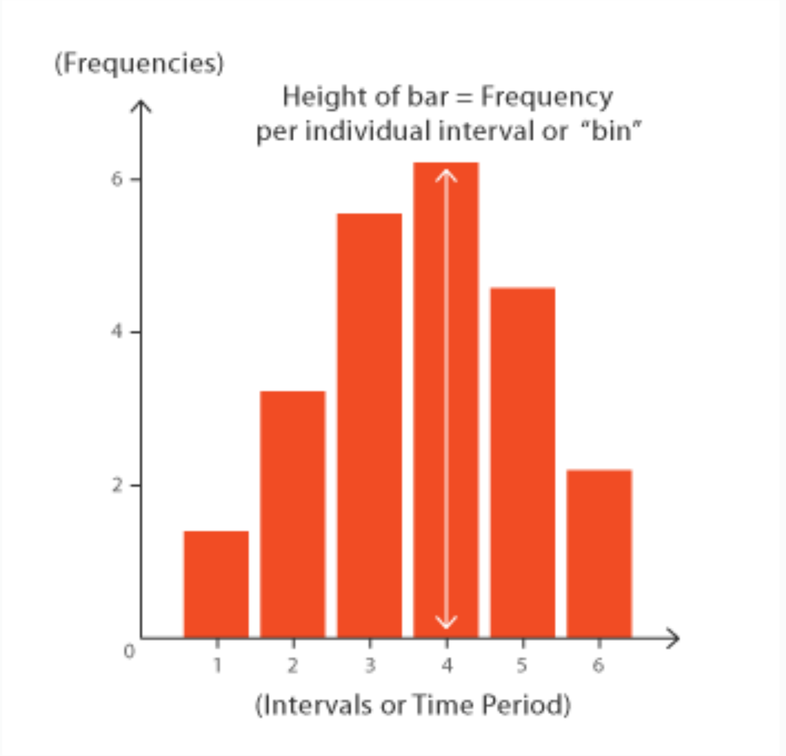
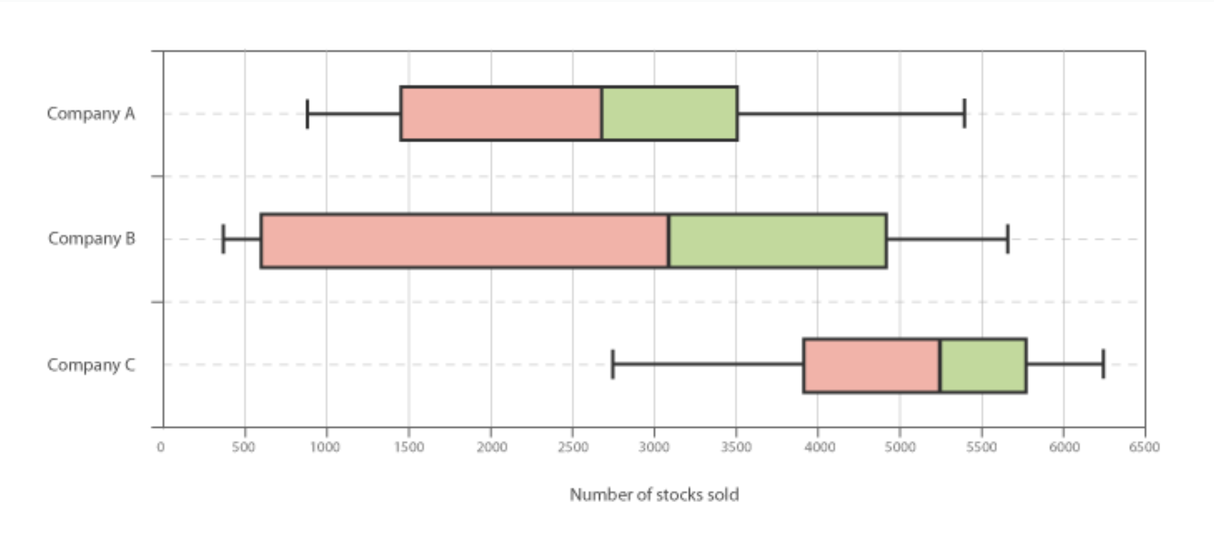


Image from Histogram, https://datavizcatalogue.com/methods/histogram.html

Box Plot



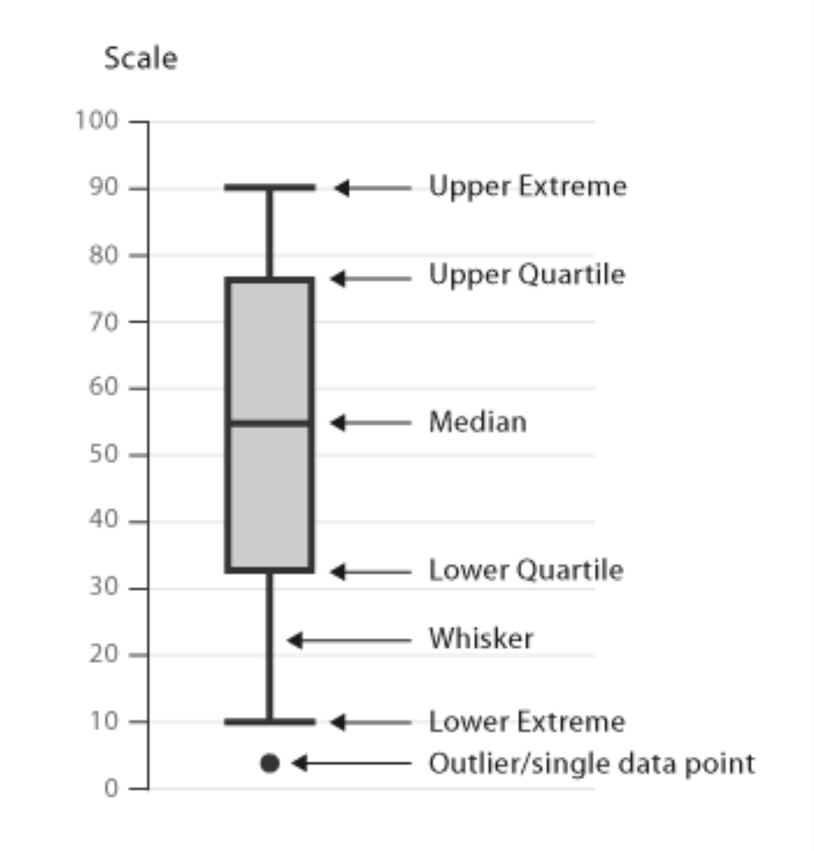
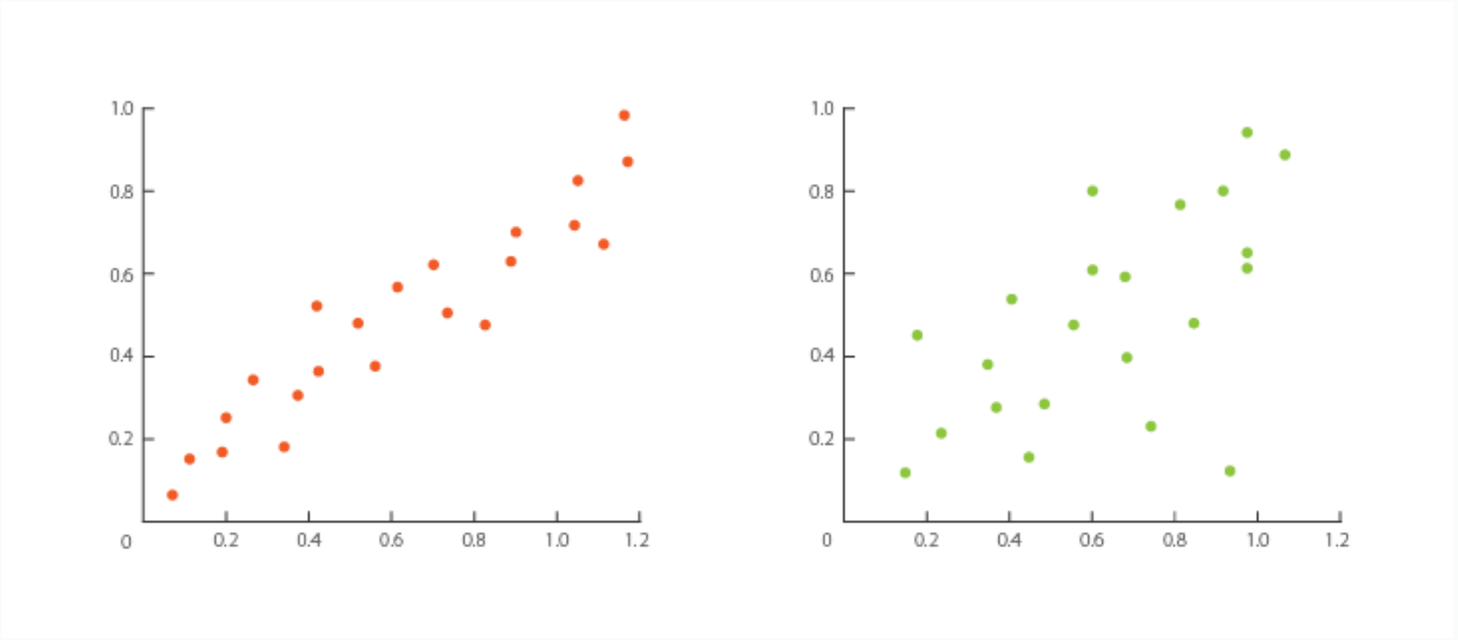


Image from Box Plot, https://datavizcatalogue.com/methods/box_plot.html

Scatter Plot



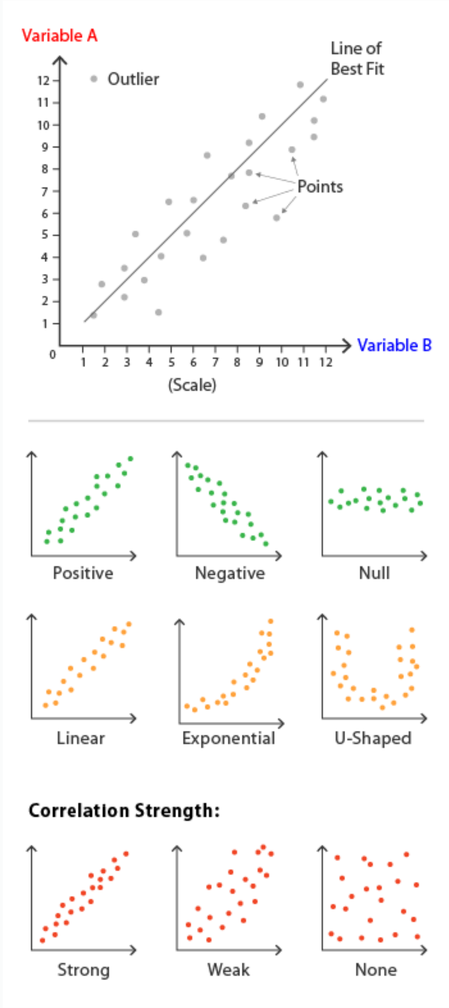


Image from Scatterplot, https://datavizcatalogue.com/methods/scatterplot.html

Word Cloud

Image from Word Cloud, https://datavizcatalogue.com/methods/wordcloud.html

Word Trees

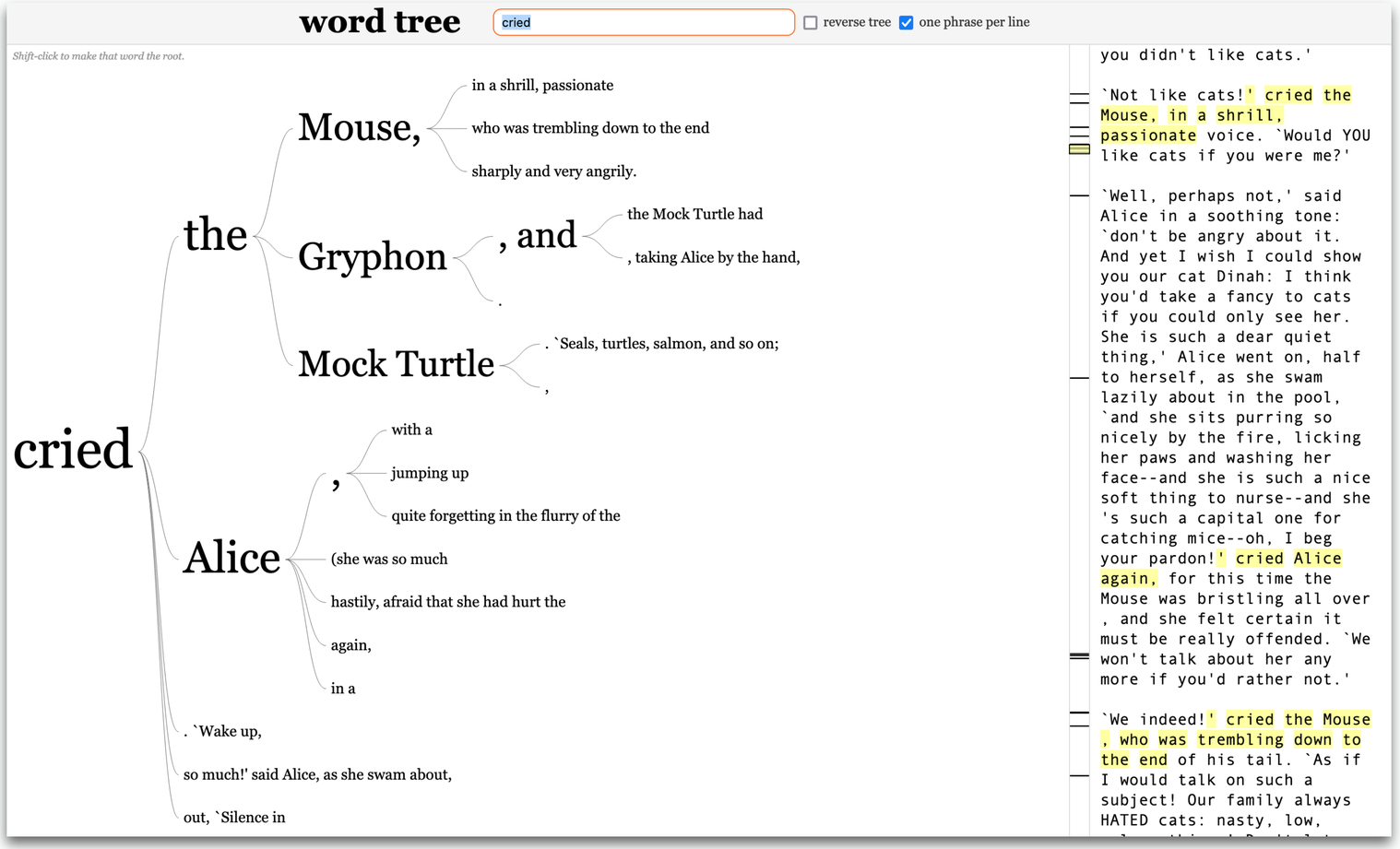


Image from Word Tree, https://www.jasondavies.com/wordtree/?source=alice-in-wonderland.txt

Google Books Ngram Viewer

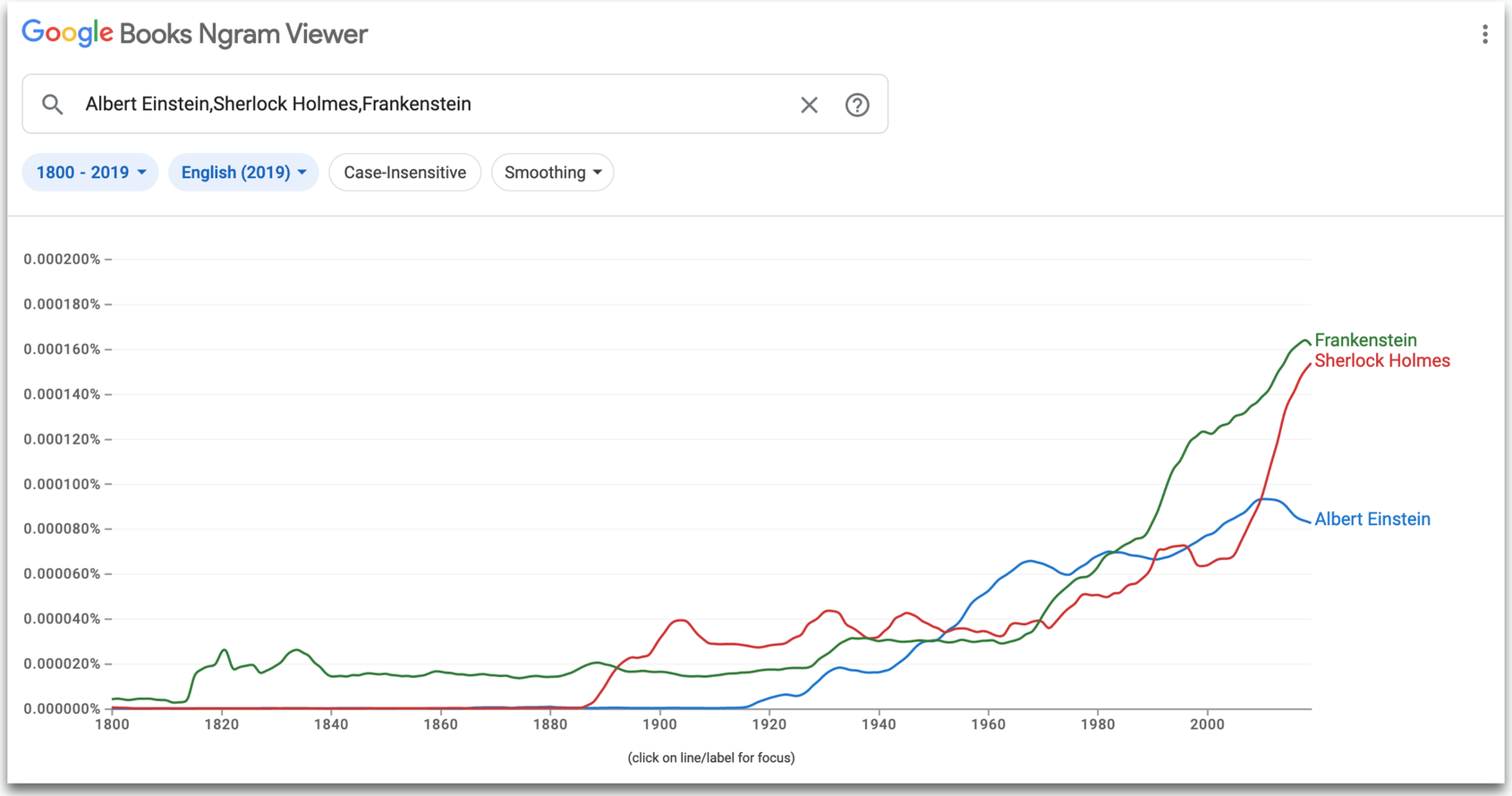


Image from Google Books Ngram Viewer, https://books.google.com/ngrams/

Tools for Data Visualization

- D3.js (Data Driven Documents), d3js.org
- Vega, vega.github.io/vega
- ggplot2 (R), ggplot2.tidyverse.org
- Bokeh (Python), bokeh.org
- p5.js, p5js.org
- Leaflet (maps), leafletjs.com

Bibliography and Further Readings

- **Designing Data-Intensive Applications** [DDIA17]
  Kleppmann, M. O'Reilly, 2017

- **Principles of Data Wrangling** [PDW17]
  Rattenbury, T., Hellerstein, J. M., Heer, J., Kandel, S., and Carreras, C. O'Reilly, 2017

- Unix™ for Poets - https://www.cs.upc.edu/~padro/Unixforpoets.pdf
  Kenneth Ward Church.

Questions or comments?