Introduction to Data Science and Data-Driven Research

In this introductory-level workshop, we will cover the basic concepts behind data science focusing on the methodological and technological aspects in working with data.

Who is the course for?

- Somebody who is not a data scientist, but is working with other data scientists.

⚙ Preparations

- No preparation is needed for this course

What is not taught?

- This course has no practical examples on how to write code for doing data science.

💬 Discussion

Let's answer to a couple of *icebreakers*

1. What are the two things about IT, programming, software, hardware, ... that you have always been curious about, but never really understood?
2. What is the most important thing that you want to learn from this module?

Episode overview

List of content:

What is data science?

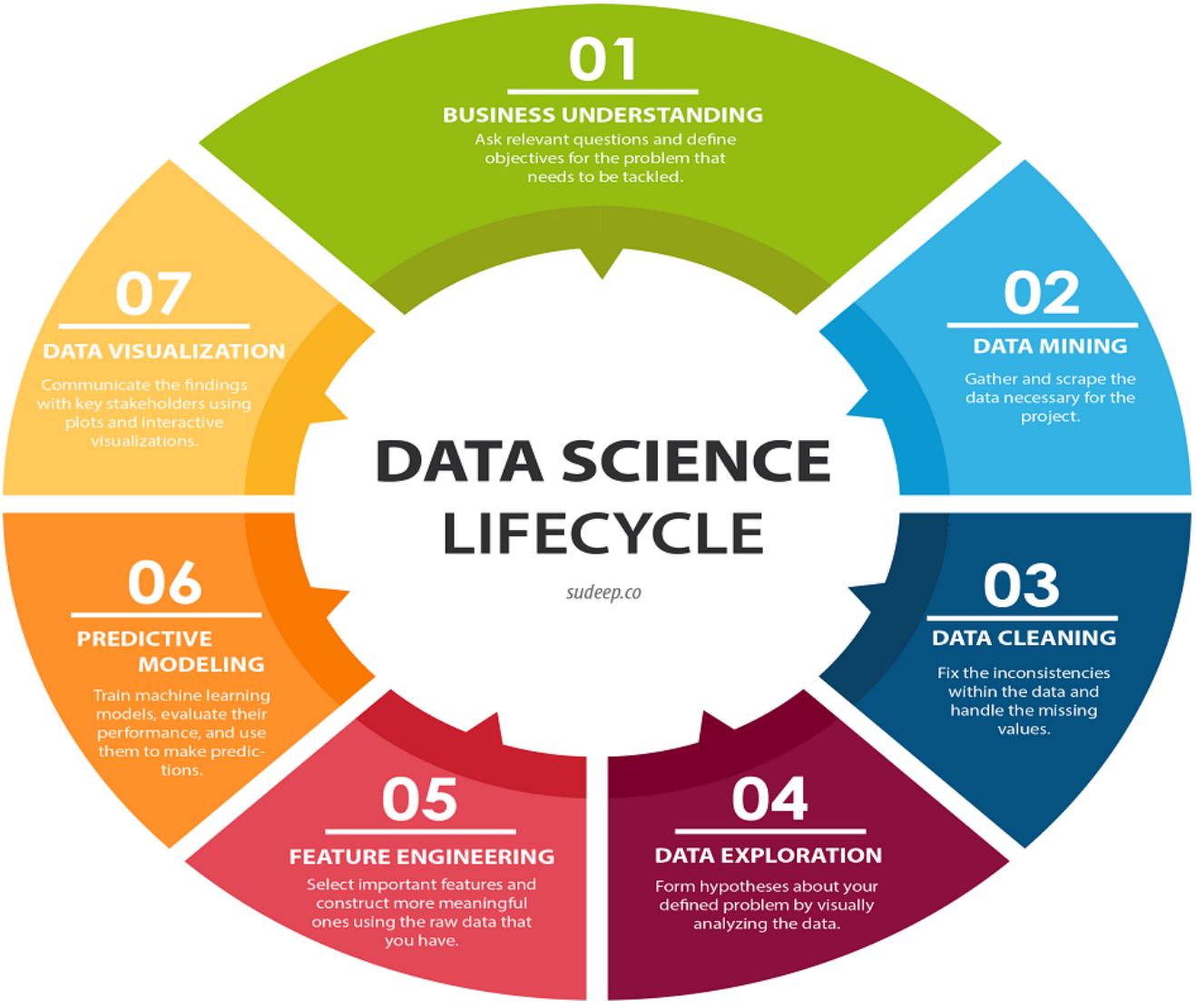
❗ Objectives

- Understanding what is meant with data science in relationship to other types of research paradigms
- Distinguishing between exploratory and confirmatory research

What is data science?

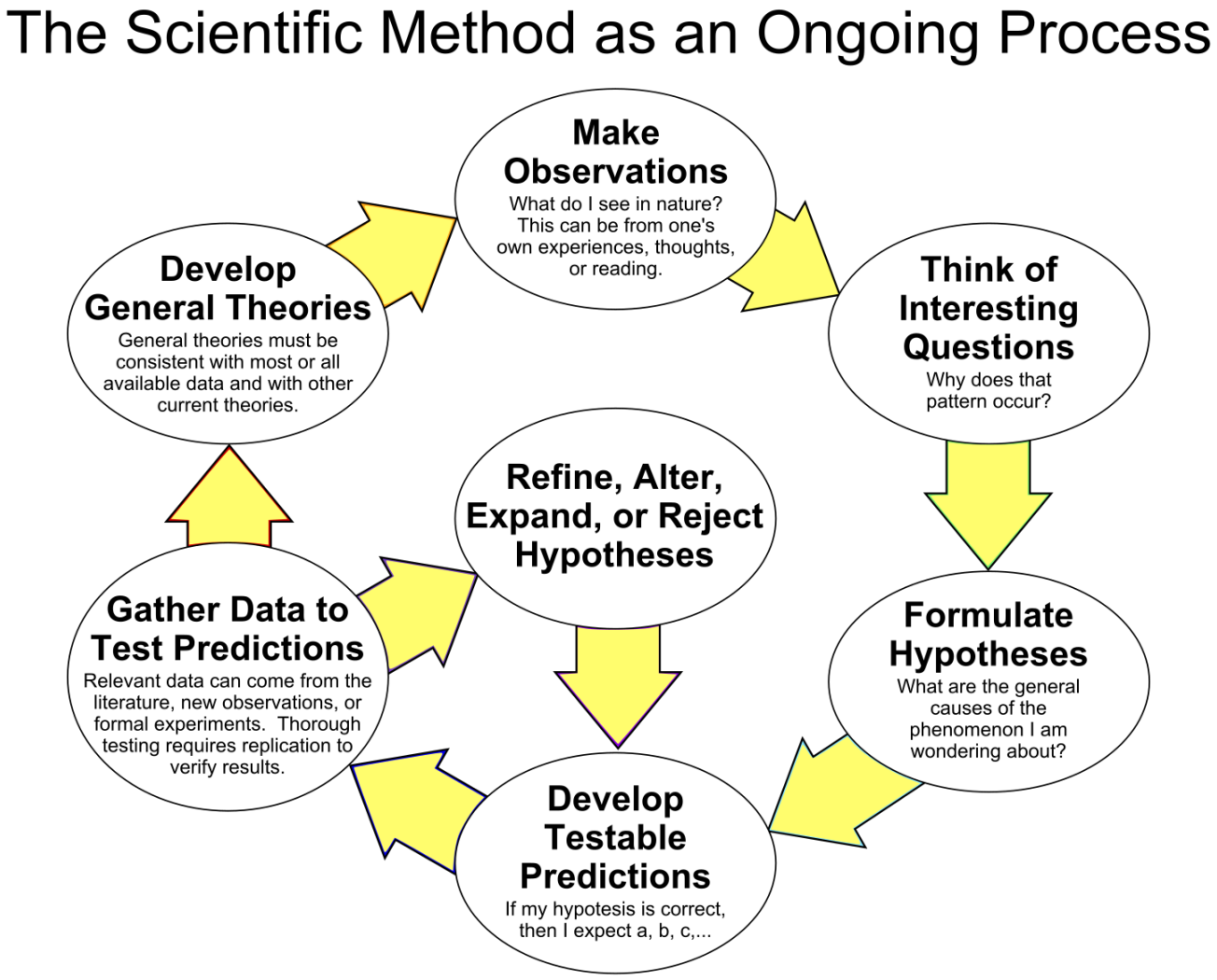
Data science is an interdisciplinary field that focuses on extracting knowledge and insights from data using methods from statistics, computer science, mathematics, and domain-specific knowledge. It involves techniques such as data cleaning, exploratory analysis, statistical testing, and machine learning to answer research questions, generate predictions, and support decision-making.

Data science is a term also used outside academia to describe those professional roles where data exploration is fundamental to gain insights about products or other metrics that a company is producing. In research, data science – also called *data-driven research* – provides a computational complement to traditional scientific inquiry. It allows researchers to work with much larger and more complex datasets than ever before.



What is the scientific method?

The scientific method is the process through which science advances, building on prior knowledge and unifying understanding over time. It played a key role in the Scientific Revolution. The method involves forming hypotheses, making predictions, and testing them through experiments. While often shown as a fixed sequence, it's better seen as a set of general principles that vary in order and application across different inquiries.



💬 Discussion

What are the similarities and the differences between data science and the scientific method?

Confirmatory (hypothesis driven) versus exploratory (data driven) research

Aspect	Exploratory Research	Hypothesis-Driven Research
Goal	Explore patterns, relationships, generate hypotheses	Test a specific, predefined hypothesis
Approach	Open-ended, data-first	Theory-first, often based on prior literature

Aspect	Exploratory Research	Hypothesis-Driven Research
Methods	Visualizations, clustering, dimension reduction, etc.	Statistical tests, regression models, experiments
Common Tools	EDA (Exploratory Data Analysis), unsupervised ML	Inferential statistics, causal inference methods
Typical Output	Insights, patterns, questions	Confirmed or rejected hypotheses

Note

Have you heard about registered reports?

> *"Which part of a research study do you believe should be beyond your control as a scientist? **The result**. Which part of a research study do you believe is most important for advancing your career? **The results**"* (Chris Chambers, slides)

Registered reports in a nutshell

(source)

- Prepare manuscript with: Introduction, Proposed Methods & Analyses, and - if applicable - Pilot data with a power analysis (Power > 90%)
- Stage 1 peer review
- **In principle acceptance** after Stage 1 peer review
- Collect and analyze data according to the plan
- Write the report (basically same Introduction and Methods, Results and Discussion are new; data and code are deposited and made accessible)
- Stage 2 peer review to confirm
- Done!
- Bonus step: include non-planned analyses (exploratory analyses)

For history about Registered reports please see Chambers & Tzavella, Nature 2021.

See also Glerean 2022 "Introduction to Registered Reports".

Summary

Data science and the scientific methods are complementary approaches in doing science with data. While traditionally a strong emphasis had been given towards quantitative sciences, the same conceptual approach applies also to qualitative research.

Note

The cooking metaphor for data science

We will often use cooking related metaphors.

Quantitative, Hypothesis-Driven

Cooking from a strict recipe: You start with a well-tested recipe (theory), follow exact steps (methods), and test if the result matches expectations (hypothesis).

Quantitative, Data-Driven (Exploratory)

Opening the fridge and experimenting: You don't have a recipe, but you've got lots of ingredients (data). You try combinations, see what patterns emerge, and maybe invent a new signature dish (model or hypothesis).

Qualitative, Theory-Driven

Get inspired before cooking: Cooking a creative dish inspired by a culinary philosophy (e.g., vegan, keto, or traditional Japanese)

Qualitative, Data-Driven

Foraging and inventing a dish: You explore the landscape (interviews, observations), gather rich ingredients (quotes, themes), and create a new dish (theory) based on what you find.

Data science on the data life-cycle

- The data lifecycle considers 7 steps in relationship to data.
- Each step however could be accompanied with choices of the methods and tools used along with the data
- Can we expand the considerations that we are familiar with from data management to methods and tools?

The Research Data Lifecycle

RDMKit defines the data lifecycle with seven stages: **Plan, Collect, Process, Analyse, Preserve, Share, Reuse**. In data science, each stage is tightly coupled with computational and methodological tasks. Each step has some questions to consider and the answers can affect reproducibility, data quality, and scientific impact:

- Plan – What data will be needed? What tools and infrastructure will be used?
- Collect – How will data be acquired or generated (e.g., sensors, experiments, scraping)? What data structure / data format will be used?
- Process – Cleaning and organizing raw data, often using programming and automation.
- Analyse – Applying statistics or machine learning to test hypotheses or make predictions.
- Preserve – Storing raw and processed data in a way that allows others (or your future self) to access and use it later.
- Share – Making data (and often code) available to others, while ensuring legal and ethical compliance.
- Reuse – Using existing data for new questions, possibly integrating it with other datasets.

Each stage in the lifecycle is computationally enabled — from writing code for data cleaning, to calculating statistics, to automating sharing via data repositories and APIs.

💬 Discussion

Explore the RDMkit website.

Spend some time with the RDMkit website: Were you already familiar with RDMkit?

✍️ Exercise

How would the software/methods life-cycle look in a field that you are familiar with?

Example: - Neuroscience: Prepare all documents for ethical review and personal data handling, collect data with the MRI scanner, move data from scanner to workstations, convert data to format for analysis, preprocess data so that statistics can be run, run statistical models (common one is the general linear model), make figures of the results (e.g. brain activation maps), share code and aggregated results on dedicated repositories (e.g. NeuroVault), make sure data can be reused (difficult one!) - *add your field / example here*

💬 Discussion

The cooking analogy for working with data and software

- Software <-> recipe
- Data <-> ingredients
- Libraries <-> pots/tools
- Operating system <-> style of kitchen we are working with
- Hardware <-> the actual kitchen with the stove, fridge, cupboards

Cooking recipe in an unfamiliar language [Midjourney, CC-BY-NC 4.0]

When we create recipes, we often use tools created by others (libraries) [Midjourney, CC-BY-NC 4.0]

Planning data science work

Planning in data science refers to the deliberate process of defining the goals, scope, data requirements, tools, methods, and risks before diving into code or analysis. It's about **designing the computational and data strategy for a research question** — and ensuring that all decisions (from data collection to modeling) are aligned with research objectives.

In research, proper planning helps:

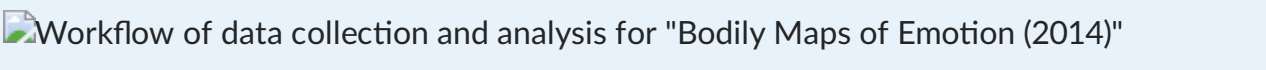
- Prevent wasted time due to poor data quality or missing variables.
- Ensure that ethical and legal responsibilities are considered before doing the actual work (e.g., data protection, consent, licenses).
- Ensure that the infrastructure can support the analysis (e.g., compute power, storage).
- Enable reproducibility and collaboration by documenting the project structure early.

Typical activities in the planning phase include:

- Defining research questions: What are you trying to discover, predict, or explain?
- Identifying data needs: Do the needed data already exist? Are new experiments needed? Are there access restrictions?
- Selecting tools and environments: Will the project use Python, R, Jupyter, cloud computing, or local servers? Will a version control system like Git be used?
- Sketching the workflow: Mapping out stages from raw data to results (sometimes visually).
- Identify who can help: colleagues, data stewards, research software engineers, IT staff, domain experts.

Note

Visualizing workflows in research papers.



From *Nummenmaa, L., Glerean, E., Hari, R., & Hietanen, J. K. (2014). Bodily maps of emotions. Proceedings of the National Academy of Sciences, 111(2), 646-651.*

Collecting data

Data collection is the process of acquiring or generating raw data that will be used in analysis or modeling. In the data science lifecycle, this step is foundational — it defines what you will be able to study, predict, and discover. Poor data collection cannot be fixed later by sophisticated models.

Data can be collected in many forms:

- Structured: Tables, spreadsheets, survey results, sensor logs.
- Unstructured: Text, images, audio, video.
- Semi-structured: JSON, XML, etc.

Why is this important in research? Without high-quality, well-documented, and ethically collected data, data science insights are unreliable. For research:

- Validity of conclusions depends on how data were gathered.
- It sets boundaries: “You can only answer the questions your data allow.”
- Poor collection leads to bias, data leakage, or irreproducibility.

The choice of file formats and data structures Sometimes we confuse file formats with data formats. It is a bit like confusing the shape of a container of a box of chocolate, with the actual shape of the chocolate pieces inside. Same file formats can contain very different data structures and what is important in computation at the end will be how the data is structured rather than if it is stored in “csv” or “xls”.

💬 Discussion

Look at the very long list of data structures in wikipedia, some data structures are “low level” and are usually masked to the final user (you do not need to know how a list is stored in a programming language). Other data structures are more general and have become part of the language used in computational sciences:

- Primitive types (fundamental when reading or writing code)
- Composite types (the classic example is a *record* which is the smallest unit of data in a database)

✍️ Exercise

Check the **tidy-data** sub-section for understanding what is a tidy dataset and how to convert data into a tidy format.

Data preprocessing

Data preprocessing is the step of **transforming raw data into a clean, consistent, and analysis-ready format**. It includes everything from fixing typos and handling missing values, to reshaping datasets and engineering new variables. Without it, models and statistics are built on shaky foundations.

Preprocessing typically involves two major tasks:

1. **Data cleaning**: Removing errors, inconsistencies, or missing entries.
2. **Feature extraction (or engineering)**: Creating useful variables or representations from raw inputs to help models or visualizations reveal insights.

Why is it needed in research?

In real-world data, messiness is the norm: - Measurements might be missing or corrupted. - Categories might be inconsistently labeled (e.g., “Finland”, “FIN”, “Suomi”). - Time data might use different formats (e.g., DD-MM-YYYY vs MM/DD/YYYY). - Unstructured data (text, images) must be structured into something a model can read.

Preprocessing ensures **validity, accuracy, and comparability**. It is often the **most time-consuming phase** in a data science project — but also one of the most critical.

Examples across disciplines:

- In **clinical studies**, missing patient records are imputed or excluded carefully to avoid bias.
- In **natural language processing (NLP)**, raw text is lowercased, tokenized, and stripped of stopwords before analysis.
- In **physics**, sensor drift or noise is corrected by calibration routines and filtering.
- In **archaeological data**, scanned artifact images are turned into tabular form by annotating features like shape or material.

Common tasks and tools:

- **Missing values**: Remove, fill (impute), or flag them.
- **Outliers**: Detect values outside expected range (statistical or domain-driven).
- **Encoding**: Turn categories into numbers (e.g., one-hot encoding for machine learning).
- **Normalization**: Scale numeric values to a standard range.

Note

Teaching and learning about data cleaning is notoriously difficult because it's not just about applying standard techniques—it's a context-dependent, messy process that demands analytical thinking, domain understanding, and hands-on experience. This article by Randy Au explains why traditional teaching methods often fall short and highlights the real-world complexity behind what may seem like a simple task.

Data analysis I: descriptive statistics and statistical testing

Data analysis refers to the methods used to **summarise, explore, and test patterns in data**. In data science, two foundational categories are:

1. **Descriptive statistics** – Describe what the data looks like.
2. **Statistical testing** – Infer whether observed patterns are likely to be real or due to chance.

Descriptive Statistics

These summarise and visualise features of the data, such as:

- **Central tendency**: Mean, median, mode.
- **Spread**: Standard deviation, range, interquartile range.
- **Shape**: Skewness, kurtosis, modality (e.g., unimodal vs. bimodal).
- **Relationships**: Correlations between variables, cross-tabulations, scatter plots.

Why this matters in research:

Descriptive statistics help researchers understand the data distribution, spot data entry errors, and guide the selection of further methods. For example:

- In **clinical studies**, summary tables show average age, blood pressure, and medication use.
- In **education research**, exam scores are summarised by class or teaching method.
- In **environmental monitoring**, daily mean temperatures and standard deviations help understand seasonal variation.

Statistical Testing

Statistical tests assess whether observed differences or relationships are likely to be **statistically significant**, given sample variability.

Some common tests:

- **T-test**: Are the means of two groups significantly different?
- **Chi-square test**: Are two categorical variables independent?
- **ANOVA**: Are there differences between three or more groups?
- **Correlation (Pearson/Spearman)**: How strong is the relationship between two variables?
- **Regression analysis**: How does one variable predict another?

Statistical testing connects your **data** to your **hypothesis**: it gives you a way to determine whether the evidence supports your theory.

✍️ Exercise

Isn't excel the most useful tool that any data- engineer | scientist | steward | manager | assistant | expert should know?

Let's learn excel by doing! Your task is as follow:

- Download the census dataset and import it in excel (let's pretend it is truly some sensitive dataset about individuals). If you do not have excel installed, Google Sheet provide a convenient web-based alternative. (Note that depending on which account you use with Google, you might not have the rights to upload real research data containing personal data.)
- Apply a filter to the data so that it is easy for you to have a look at the values in each column
- Data cleaning: This data has way too much personal identifiers, not really needed for research! Remove (or mark in some way) those columns that can be used to directly identify an individual.
- Descriptive statistics I : plot an histograms of the column Age
- Optional - Descriptive statistics II: plot an histograms of the column Age so that exact ages are show in the X axis
- Descriptive statistics III: calculate the average of the column Age
- Export/download/save the excel file (only the sheet you worked on) and submit it as homework.
- Optional task (difficult!): do you think this cleaned version of the dataset could be shared openly? Do you think the histogram with exact ages in X axis could be included in a paper?
- Optional task (very difficult): What would you do to make sure there are at least 5 individuals with the same age?

Beyond the basics:

- **Effect sizes and confidence intervals**: A statistically significant result ($p < 0.05$) might still be unimportant. Always report effect sizes and uncertainty intervals.
- **Multiple comparisons**: Testing dozens of variables increases the chance of false positives. Use corrections (e.g., Bonferroni, False Discovery Rate) to control error rates.
- **Assumptions matter**: Many tests assume normal distributions or equal variances. When those don't hold, non-parametric alternatives (e.g., Mann–Whitney U test) are safer.
- **Visual diagnostics**: Always pair numerical tests with plots — histograms, box plots, scatter matrices — to catch anomalies or misinterpretations.

Exploratory vs. confirmatory: Exploratory data analysis (EDA) looks for patterns. Confirmatory analysis tests predefined hypotheses. Mixing them without care can lead to bias.

Data analysis II: artificial intelligence and machine learning

What is machine learning (ML) and artificial intelligence (AI)?

- **Artificial Intelligence (AI)** refers to systems that mimic human cognitive functions like learning, reasoning, and problem-solving.
- **Machine Learning (ML)** is a subset of AI focused on systems that learn from data — improving their performance on a task over time **without being explicitly programmed**.

In a research context, ML is a powerful tool for:

- **Pattern recognition** (e.g., detecting disease in medical images),
- **Prediction** (e.g., forecasting species extinction risk),
- **Classification** (e.g., distinguishing fake news from real news),
- **Clustering** (e.g., grouping similar genetic expressions),
- **Recommendation** (e.g., suggesting relevant articles or treatment plans).

Unlike traditional statistical testing, which relies on predefined hypotheses and assumptions, machine learning is often **data-driven, inductive**, and focused on **prediction** or **automation** rather than explanation.

Types of Machine Learning

Supervised learning (training with labeled data):

- **Classification**: Predict categories (e.g., spam vs. not spam).
- **Regression**: Predict numerical outcomes (e.g., house prices).

Unsupervised learning (no labels, discovering structure):

- **Clustering**: Group similar data points (e.g., market segmentation).
- **Dimensionality reduction**: Simplify complex datasets (e.g., PCA).

Examples across disciplines:

- **Healthcare**: Predicting patient readmission based on historical data.
- **Astronomy**: Classifying galaxy types from images.
- **Literary studies**: Using NLP to identify themes or authorship in large text corpora.

Ecology: Modeling animal movements based on GPS and environmental data.

Note

How do large language models work? How can I use them responsibly in my work?

A comprehensive introduction on the topic with focus on responsible conduct of research is available at this zenodo link and as video at Aalto University Research Services YouTube channel

Data analysis III: qualitative research methods

Qualitative analysis often involves methods like **deductive and inductive coding**. Deductive coding starts with pre-defined categories or theories and applies them to the data, while inductive coding allows themes to emerge naturally from the data itself without prior assumptions. Both approaches help researchers identify patterns, meanings, and insights within textual or observational data, and can also be combined in a flexible, iterative process.

The process for working with qualitative research methods is usually less automated and *automatable* when we compare it with quantitative methods. Recent developements in Natural Language Processing have added the possibility to also automate the coding process, however they might defeat the purpose of qualitative research if the coding is reduced to basic textual classification. It is a currently debated topic in the field of qualitative research (more references coming).

Data visualisation techniques

Data visualization is the practice of **translating data into visual formats** — such as charts, graphs, and maps — to help explore, understand, and communicate patterns and insights. In the data science lifecycle, visualization supports **both analysis and *storytelling***.

- During analysis: used to explore distributions, detect anomalies, and observe relationships.
- During communication: used to present findings to technical and non-technical audiences.

Why is it important in research?

Visualization is essential to:

- Reveal **trends and outliers** that raw data or statistics may obscure.
- Support **hypothesis generation** in exploratory analysis.
- Communicate **complex findings clearly** to collaborators, funders, and the public.
- Enhance reproducibility — well-documented plots help others verify your results.

Common types of visualizations:

- **Exploratory**:
  - Histograms and boxplots (distribution),
  - Scatter plots and heatmaps (relationships),

    - Line plots (trends over time),
    - PCA plots (dimensionality reduction).
- **Explanatory**:
    - Bar charts, annotated maps, Sankey diagrams, interactive dashboards.

✍ Exercise

Explore all the possible type of plots available at https://r-graph-gallery.com/

- Was there a type of plot that you were not familiar with?
- Did you know that some types of plots can be more biased than others and wrongly lead to conclusions that are actually not supported by the data? (see for example "Beyond Bar and Line Graphs: Time for a New Data Presentation Paradigm")

Preserve, Share, Reuse

This stage of the data lifecycle focuses on ensuring that datasets, code, models, and insights from a research project **do not disappear after publication**, but remain accessible and usable for future projects — by yourself, your team, or the broader community.

These steps are closely aligned with the **FAIR principles**:

- **Findable**: Data and metadata can be located by others.
- **Accessible**: Data is available under clear conditions.
- **Interoperable**: Data can be integrated with other systems.
- **Reusable**: Data is well-described and licensed for reuse.

In data science, "preservation" extends beyond raw data:

- Preprocessed/cleaned data,
- Code used for analysis,
- Trained models and configurations,
- Logs, metadata, and computational environments (e.g. container images).

Why is this important in research?

- Enables **replication and verification** of results (scientific integrity).
- Allows others to **build upon your work** (cumulative knowledge).
- Satisfies requirements of funders and publishers.
- Increases **impact and citations**.
- Saves time: you or your team can reuse workflows or cleaned datasets in future studies.

Common tools and repositories:

- **Data repositories**: Zenodo, Figshare, Dryad, Dataverse, OpenAIRE.
- **Code repositories**: GitHub, GitLab (with README.md, license, and citations).

- **Notebooks**: Jupyter/Quarto projects saved with environment metadata (requirements.txt, environment.yml).
- **Containerization**: Docker/Singularity images to preserve computing environments.

Model documentation: Model cards (for models) and datasheets (for datasets).

Beyond the basics:

- **Metadata and documentation**:
  - Raw data is rarely useful without context. Metadata standards (like Dublin Core, DataCite, or domain-specific schemas) make data understandable and reusable.
  - Good README files, manifest files, and inline code comments make preservation valuable.
- **Versioning**:
  - Use Git (or DVC for data) to track changes. Without versioning, it's hard to know what version of the data or model a result came from.
- **Licensing**:
  - Reuse depends on legal clarity. Use open licenses (e.g., CC BY 4.0 for data, MIT or GPL for code) to allow others to build on your work.
  - Ensure you have the rights to share what you publish (especially for collaborative or sensitive datasets).
- **Sensitive data and controlled access**:
  - For personal data (e.g., patient records, interviews), open sharing is restricted. However, preservation is still essential — use **Trusted Research Environments** or controlled-access repositories with Data Use Agreements (DUAs).
- **Persistent identifiers and citation**:
  - Use DOIs for datasets and models to make them citable.
  - Adopt standards like the Joint Declaration of Data Citation Principles.
- **Model and dataset governance**:
  - AI systems require monitoring post-publication. Future stewards might need to update or "unlearn" parts of models if biases or errors are discovered.
  - Consider maintaining "model sheets" with version info, training data details, and known limitations.

Building a Supercomputer

Warning

This material is **work in progress**.

- Writing technical materials as a technical person falls into the fallacy of assuming the reader shares the same background knowledge, leading to explanations that skip essential context, use jargon without clarification, and unintentionally exclude non-expert audiences. **Please ask to expand and explain obscure concepts, jargon, unclear things**

- It is difficult to find CC0 graphics to reuse so the materials right now are not too visual. Please suggest if you have any good source for graphics.

In this episode, we try to build a supercomputer — starting from a single computer and scaling it up, step by step. Along the way, we'll see why researchers use powerful computers and how the infrastructure behind research computing actually works.

Motivation: Why Do We Need Computers?

Computers have become essential to research in every field — from physics to philosophy. But why?

The core reasons are speed, automation, and the ability to **store, process, and analyze large amounts of information** quickly and reproducibly. Researchers use computers to perform tasks that would otherwise take months or years to do by hand — or would be too complex to do at all.

💬 Discussion

Why Do Researchers Use Computers?

Let's look at a few examples:

- **In genomics**, computers process the DNA sequences of thousands of individuals, looking for patterns that might explain disease.
- **In climate science**, computers simulate Earth's atmosphere and oceans to forecast climate change scenarios decades into the future.
- **In linguistics**, researchers use natural language processing (NLP) to analyze the evolution of meaning in massive text archives across centuries.
- **In the humanities**, historians use text mining to analyze themes across thousands of digitized books.
- **In business research**, analysts model customer behavior using time series and recommendation algorithms.

Without computers, these projects would either be **impossible** or so slow that the results would arrive too late to be useful.

Automation, Reproducibility, and Exploration

Computers allow researchers to:

- **Automate repetitive tasks** (e.g. cleaning 10,000 images),
- **Try many different models quickly** (e.g. training 100 regressions with different parameters),

- **Ensure reproducibility**: code and scripts can be reused, shared, and rerun by others with identical results.

This is especially important in modern science, which increasingly depends on **data-driven inquiry** and **transparent methods**.

💬 Discussion

Cooking Metaphor: Computers Are Research Kitchens

Imagine a researcher as a cook, and their ideas as recipes.

- Without a kitchen (computer), the cook is left to chop and stir with bare hands. It's slow and exhausting.
- With a modern kitchen, tools like ovens (CPUs), food processors (GPUs), and timers (scripts) make cooking fast, repeatable, and precise.
- Just like kitchens, **computers turn raw ingredients (data)** into **finished meals (results)** — following detailed recipes (programs and workflows).

Different research domains need different kitchens: a historian may just need a stovetop, while a physicist might need an industrial food lab. But **everyone benefits from the tools a computer provides.**

Bonus Insight

Even though researchers come from very different backgrounds, the **basic functions computers provide are universal**:

- Input: Collect or read data,
- Processing: Apply algorithms, filters, or simulations,
- Output: Store, visualize, or share results.

The tools vary — spreadsheets, R scripts, Python notebooks, cloud platforms — but the principles are the same. Learning how to think about computing **systematically and metaphorically** is a first step to becoming a confident data steward.

Motivation: Do I Need a Supercomputer?

Not all research requires a supercomputer — but some does. Whether you need one depends on **what kind of questions you're asking**, **how much data you're working with**, and **how fast you need results**.

When Is a Normal Computer Enough?

For many research tasks, a **well-equipped laptop or desktop** is completely sufficient:

- Analyzing a survey dataset in SPSS or R.

- Writing scripts in Python to clean tabular data.
- Creating visualizations or writing reports.

These are like home-cooked meals — you only need **one cook** in a regular kitchen.

When Do You Need More Power?

Some research tasks **can be split into smaller pieces** and run in parallel — like having **many chefs working at once**, each preparing one part of the meal.

This is where **supercomputers** come in:

- **Molecular simulations**: Thousands of atoms simulated over millions of time steps.
- **Climate modeling**: Regional and global forecasts require solving differential equations for every square kilometer of the Earth's surface.
- **AI training**: Large language models (LLMs) like GPT-4 require thousands of GPU hours to train.
- **Satellite image processing**: Terabytes of high-resolution images analyzed over large time windows.

Supercomputers allow these tasks to be done **in hours or days instead of months or years** — and often, to be done at all.

💬 Discussion

Cooking Metaphor: Do You Need an Industrial Kitchen?

Imagine you are making:

- One bowl of pasta for yourself → **your laptop is fine**.
- 100 bowls of different pastas for a party → **you need help**, maybe some automation.
- Tens of thousands of portions for a food festival → You need **an industrial kitchen with many chefs working in parallel**, all coordinated.

That's what a supercomputer is: **many normal computers working together**, connected with high-speed communication, and orchestrated to act like one large kitchen.

Not All Remote Computers Are Supercomputers

A supercomputer is powerful, but sometimes you just need **a different computer**, not a "super" one.

For example:

- A **remote workstation** at your department could be enough for heavier tasks
- A **server** with more RAM or a better GPU than your laptop, usually on-premises.

- A **remote virtual machine (VM)** hosted in the cloud (e.g. on CSC, AWS, Azure). Servers can also be virtual machines, from the user perspective nothing changes

These can help you:

- Work with large datasets that don't fit on your laptop,
- Share resources with collaborators,
- Access faster networks or licensed software.

Remote ≠ Super. But remote access is still incredibly powerful.

💬 Discussion

How to know when I need to scale up to a bigger system?

Understanding what you need to solve your computing problems is not trivial. Some questions you might ask yourself at some point in your life:

- **Do I need one or more CPUs to process my data?** *Caveat: not every problem can be parallelised.*
- **Do I need 1 computer with N CPUs or N computers with 1 CPU?** *Caveat: not all code is written to use multiple CPUs*
- **Do I need GPUs?** *Caveat: GPUs require special code/libraries*
- **Do I have tools/code that can be run on multiple CPUs and/or GPUs?** *Caveat: code does not automatically use all the resources you might have at hand*
- **Is the speed to access the data a bottleneck?** *Caveat: the faster the access to the data, the more localised your data is*
- **What level of security my data (and code) need to have?** *Caveat: sensitive data should not be taken outside remote storage locations*

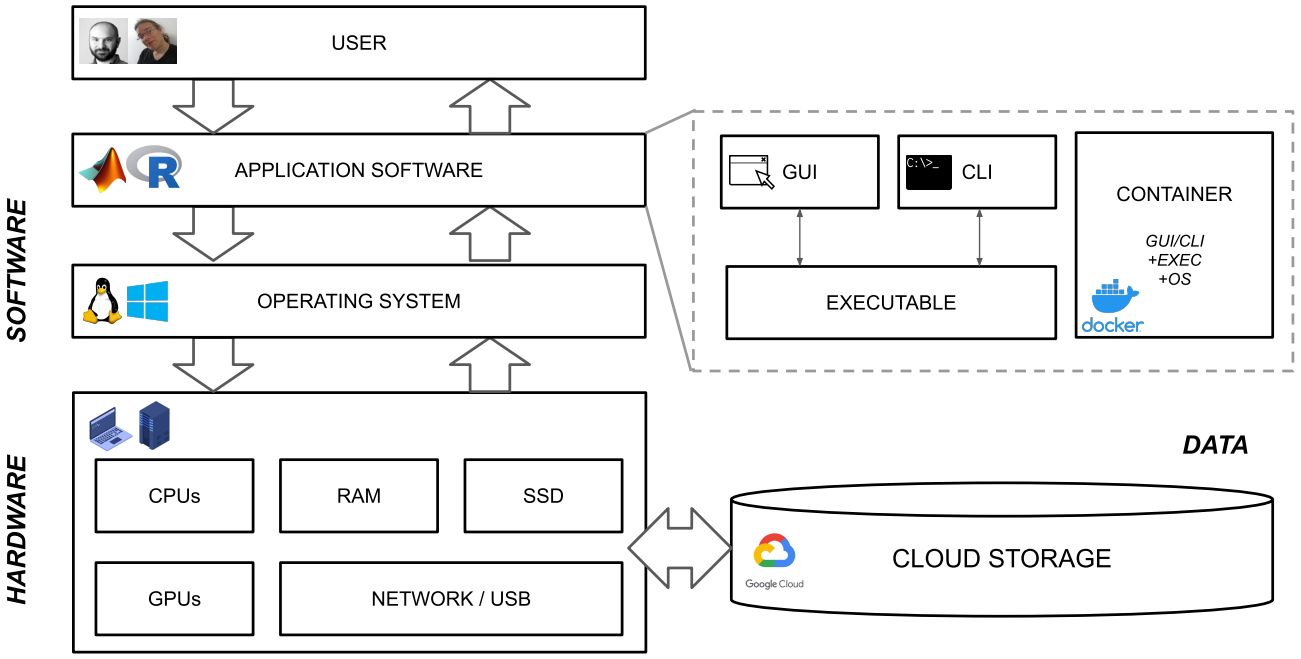
Key Takeaway for support teams

Research software engineers along with data stewards can help researchers **decide where to run what** with the perspective of the copmuting needed and the data usage. This includes:

- Estimating whether a task needs local or remote computing (e.g. for using larger storage or more secure storage),
- Helping select platforms (e.g. supercomputer, cloud VM, remote workstation),
- Explaining trade-offs (e.g. cost, speed, access, learning curve),
- Ensuring **ethical and legal use** (e.g. personal data on secure servers only).

Understanding the **computational scale of the problem** is a critical support skill.

Let's build a supercomputer then!



The architecture of a computer (or computing node)

Step 1: Build a Computer

Let's start from scratch. Before we can build a supercomputer, we need to understand how a **regular computer is made** — and what each component does.

A modern computer is a collection of hardware pieces that work together to process data, store information, and interact with users.

Essential Hardware Components

CPU (Central Processing Unit)

- Often called the *brain* of the computer.
- Executes instructions from programs, one after another or across multiple cores.
- Controls the flow of operations — calculations, logic, and decision-making.

GPU (Graphics Processing Unit)

- Originally for graphics, now essential for **parallel computing**.
- Contains many cores optimized for similar repetitive tasks — such as training AI models or processing thousands of images.
- Useful in research tasks that involve **matrix operations** or **large-scale data parallelism**.

Note

What is the difference between CPU and GPU?

A short video that intuitively explains the difference between CPUs and GPUs

RAM (Random Access Memory)

- Temporary, fast memory used while your programs are running.
- Holds **active data** and **code in use**.
- Not the same as storage! RAM is cleared when the computer is turned off.

Note

RAM is not the only memory in the computer

While **RAM (Random Access Memory)** is the most well-known type of memory, it's just one part of a broader memory system that computers use to operate efficiently. Other important types include:

- **Cache memory**: Small but extremely fast memory located on or near the CPU. It stores frequently accessed instructions and data to speed up execution.
- **Registers**: Tiny storage units inside the CPU used to hold data that's actively being processed. They operate faster than any other memory type.
- **ROM (Read-Only Memory)**: A type of non-volatile memory that stores permanent instructions, such as the computer's firmware (e.g., BIOS or UEFI), which runs during startup.
- **Virtual memory**: A portion of storage (e.g., your hard drive or SSD) that the operating system uses to simulate additional RAM when the actual RAM is full. It's much slower but expands available working space.
- **Swap space**: A specific part of the disk reserved by the OS (especially in Linux) to support virtual memory by temporarily holding data from RAM.
- **VRAM (Video RAM)**: Memory used by GPUs (Graphics Processing Units) to store visual data like textures, 3D models, or image matrices. It's optimized for parallel processing and is essential for high-performance tasks such as AI model training and image rendering.

Each of these memory types serves a specialized function. Together, they ensure that different parts of the computer — from the CPU to the GPU — have quick access to the data they need.

Storage (HDD/SSD)

- Long-term storage of files and programs.
- SSD (Solid State Drive) is faster and more common in research laptops than HDD.
- Stores your operating system, datasets, documents, and applications.

Note

Not all storages are the same

Not all storage devices perform the same — they differ in **speed**, **reliability**, and **technology**.

- **SSD (Solid State Drive)**: Fast, reliable, and now common in laptops and servers. Uses flash memory with no moving parts, which makes it much faster than older technologies.
- **HDD (Hard Disk Drive)**: Slower and more prone to wear because it uses spinning magnetic disks and a moving read/write head. Still used for large-capacity archival storage due to lower cost per terabyte.
- **eMMC (embedded MultiMediaCard)**: Slower flash storage found in budget devices like tablets or low-end laptops. Not designed for heavy workloads.
- **Optical drives (CD/DVD)** and **magnetic tape**: Rare today in personal computing, but still used for long-term cold storage or archival backups. Very slow access speeds.

In modern research workflows, **SSD is preferred** for active data processing because of its high read/write speeds, while HDDs or network-attached storage may be used for backup or long-term storage.

Network Interface Card (NIC)

- Connects your computer to the internet or local network.
- Can be wired (Ethernet) or wireless (Wi-Fi).
- Essential for remote access, cloud computing, and downloading datasets.

Note

The Internet, IP addresses, Virtual Private Networks, ssh

The **Internet** is a global system of interconnected networks that allows computers to communicate using standardized protocols. It underlies many services — not just web browsing, but also email, file sharing, and secure access to remote machines.

Every device connected to the internet has an **IP address** — a unique identifier, like a mailing address, that allows data to be sent to and from the correct location. Some IP addresses are **static** (they never change), while others are **dynamic** (assigned temporarily by your internet provider and may change when you reconnect). Servers usually have static IPs so they can be reached reliably.

To protect internal services, many universities and institutions **restrict access based on IP addresses**. If you're working remotely, you may not be allowed to connect directly to internal systems. This is where a **Virtual Private Network (VPN)** is useful. A VPN creates a secure, encrypted tunnel between your computer and your institution's network. When connected through a VPN, your computer appears as if it is inside the university — which enables access to licensed resources, shared drives, internal portals, or computing environments that would otherwise be blocked.

A key tool for accessing remote research infrastructure is **SSH (Secure Shell)**. SSH allows you to securely log into another computer or server using only a terminal window — no graphical interface is needed. It provides command-line access to powerful systems like cloud servers or supercomputers. SSH is encrypted and can be secured with **cryptographic key pairs**, making it both safe and widely trusted in scientific computing.

In practice, support teams and researchers may use **SSH in combination with VPNs** to connect to institutional clusters, manage datasets, and run computational workflows from anywhere in the world. Firewalls and institutional policies often limit who can access what, so access may require registration or setup in advance.

Note

Different networks – different speeds

Not all internet connections are equally fast — and the type of connection you use can significantly affect **how quickly you can access cloud storage or transfer data**.

- **Wired Ethernet** (e.g., Gigabit Ethernet): Typically offers stable, high-speed connections around **1 Gbps (1000 Mbps)**. Ideal for downloading large datasets or syncing with remote storage.
- **Wi-Fi**:
  - **Wi-Fi 5 (802.11ac)**: Up to **400–900 Mbps** in real conditions.
  - **Wi-Fi 6 (802.11ax)**: Can reach **1–2 Gbps**, but speed varies with distance and interference.
- **Mobile Networks**:
  - **4G**: Real-world speeds are around **20–100 Mbps** — enough for email or browsing, but slow for large file transfers.
  - **5G**: Can offer **200 Mbps to over 1 Gbps** depending on coverage — fast enough for cloud access, but latency and reliability may vary.

When using cloud storage or remote servers, a **slow connection can become a bottleneck** — even if your storage or computing resources are fast. For large data transfers, **wired Ethernet is usually the most stable and efficient**.

Power Supply Unit (PSU)

- Converts electricity from the wall into usable power for your components.

Motherboard

- The main circuit board where all components plug in.
- Handles communication between the CPU, RAM, storage, and peripherals.

Interfaces for the User

These are the parts that make a computer usable by humans:

- **Monitor** (screen): to see the output.
- **Keyboard & mouse**: to provide input.
- **USB/HDMI ports**: for connecting external devices (storage drives, cameras, projectors).

These are not just conveniences — **they're essential for interacting with the system**, especially during setup or troubleshooting.

💬 Discussion

Cooking Metaphor: Assembling Your Research Kitchen

- **CPU** = the chef: decides what to cook and when.
- **GPU** = food processor: powerful but specialized.
- **RAM** = kitchen counter: workspace where prep happens.
- **Storage** = pantry and fridge: long-term ingredient storage.
- **NIC** = the telephone or delivery door: communicates with the outside world.
- **Motherboard** = the kitchen layout: connects all the tools.
- **Keyboard and mouse** = your hands and tools: how you work in the kitchen.

Without these components in place and connected, your kitchen — or your computer — can't function.

Bonus insight for support teams

Support teams often help researchers choose:

- Which **hardware specs** are needed for a task,
- Whether a local machine is sufficient,
- Or whether a shared or remote resource is more appropriate.

Understanding these components helps you:

- Read tech specs with confidence,
- Talk with IT teams or vendors,
- Support researchers with realistic expectations about performance.

Step 2: Add Life to the Hardware

So far, we've assembled the physical components of a computer — the *kitchen* is ready. But a pile of hardware isn't useful on its own. We need to make it come alive.

This is where **firmware** and **boot systems** come into play.

🧬 What Is BIOS (or UEFI)?

The **BIOS** (Basic Input/Output System) is a small program stored on a chip on the motherboard. It runs **immediately when the computer is powered on** — before any operating system is loaded.

Modern systems often use **UEFI** (Unified Extensible Firmware Interface), which is an advanced version of BIOS.

What Does the BIOS/UEFI Do?

1. **Power-On Self-Test (POST)**: Checks that components like RAM and keyboard are working.
2. **Initial Setup**: Detects connected drives and peripherals.
3. **Boot Order**: Decides where to look for the system to load next (e.g., SSD, USB stick, network).
4. **Hands Off**: Once a bootloader is found, the BIOS/UEFI gives control to it.

In short: The BIOS is the spark that tells the hardware to wake up and start the process of becoming a computer.

Is BIOS Enough? No — We Need an Operating System

The BIOS only prepares the computer to load a **real operating system** (like Linux, Windows, or macOS). Without an OS, your computer can't:

- Launch programs,
- Manage files,
- Connect to the internet,
- Display a graphical interface.

We'll install the OS in the next step — but first, let's talk about early-stage **security**.

🔐 Security Already Starts Here: Disk Encryption

At the BIOS/UEFI level, we can:

- Set a password to **prevent unauthorized access to the firmware settings**,
- Enable **disk encryption** to protect data at rest.

Encryption scrambles your data using a mathematical key. Without that key, the data is unreadable — even if someone physically steals the disk.

Think of encryption like a secret spice blend locked in a safe: even if someone gets the ingredients (data), they can't cook the same recipe without the key (password or passphrase).

> 💬 Discussion
>
> **Cooking Metaphor: Lighting the Stove**
>
> - The BIOS is like **turning on the stove and checking the gas** — it prepares everything to start cooking.
> - It doesn't do any cooking itself — but without it, the kitchen is dark.
> - Encryption at this stage is like **locking your ingredient cabinet**, so no one can mess with it while you're away.

Bonus Insight for support teams

Why this matters:

- Researchers working with **sensitive data** (e.g., health, interviews, unpublished results) must **protect their devices** from theft or unauthorized access.
- As a support person, you may be asked how to:
  - Enable disk encryption,
  - Set up secure boot settings,
  - Ensure laptops comply with institutional data protection policies.

Understanding BIOS/UEFI is not about being a technician — it's about knowing **where trust begins** in the system.

Step 3: Add Usability with the Operating System

You've assembled the computer. You've powered it on. The BIOS did its job — and now it's time to give the machine a personality: an **operating system**.

Without one, the computer is like a kitchen with gas and electricity but **no recipes**, **no tools**, and **no staff**. It's technically on, but it doesn't *do* anything.

What Is an Operating System (OS)?

An operating system is software that:

- **Manages hardware**: Memory, disk, CPU, and input/output devices.
- **Runs programs**: Coordinates which code executes and when.
- **Controls user access**: Determines who can do what.
- **Provides interfaces**: Command line, windows, files, folders, menus.

Examples:

- **macOS** (popular in creative and academic work).
- **Windows** (common for desktops),
- **Linux** (used widely in research environments and HPC),

💬 Kitchen analogy

Just for fun: which operating systems do the following example kitchens represent?

1 2 3

[Midjourney, CC-BY-NC 4.0]

How Programs Work With the OS

- When you open a program, the OS loads it into **RAM**.
- The CPU executes the program **instruction by instruction**.
- The OS handles **scheduling**: when each program gets CPU time.
- The OS also provides **system calls** so that programs can talk to devices (e.g., save a file, read a sensor).

So the OS is the **bridge** between raw hardware and usable computing.

Users and Roles in the OS

Operating systems allow **multiple users**. Each user may have different permissions.

- **Root / Administrator**:
  - Full control of the system.
  - Can install software, change user accounts, reconfigure everything.
- **Normal Users**:
  - Limited control — can run software, manage their own files, but not others'.

This separation prevents accidents or security issues (e.g. if you run a malicious program that deletes the main operating system files, user permissions wouldn't let you do that).

Security and Permissions

The OS uses **file permissions** and **user roles** to:

- Ensure data integrity (users can't modify each other's files),
- Maintain confidentiality (research data is private),
- Protect system functionality.

Permissions control:

- Who can read, write, or execute files.
- Who can install or update software.
- Which users can access shared data folders.

Permissions are like **kitchen access badges** — only the head chef can restock the fridge or replace knives, while sous-chefs can use them but not reorder.

Note

Permissions in the Linux terminal explained

When you list files in a Linux terminal using the command `ls -l`, you'll see something like this:

```
-rw-r--r--  1 user group  2048 Apr  5 14:32 data.txt
```

Let's break it down:

- The first part, `-rw-r--r--`, shows the **file permissions**.
- It's a 10-character string where:

- The first character indicates the **type** (`-` = regular file, `d` = directory, `l` = symbolic link, etc.).
- The next 9 characters are split into **three groups of three**:
  - `rw-`: **Owner** permissions (read, write),
  - `r--`: **Group** permissions (read only),
  - `r--`: **Others** (everyone else; read only).

So in this example:

- The **owner** can **read and write** the file,
- The **group** can only **read** it,
- **Others** (any user on the system) can also only **read** it.

If you see something like this for a script:

```
-rwxr-xr-x
```

That means:

- The owner can **read, write, and execute**,
- The group and others can **read and execute**, but **not modify** the file.

You can change these permissions using commands like `chmod`, and change the owner with `chown`.

Understanding these settings is essential when managing access to data and scripts in shared research environments.

💬 Discussion

Cooking Metaphor: The Operating System Is the Kitchen Manager

- The OS decides who can use which appliances.
- It makes sure the oven isn't on fire while the dishwasher is running.
- It keeps the pantry organized (file system),
- And ensures everyone follows safety protocols (permissions and users).

Without an OS, you'd just have an empty room full of disconnected appliances.

❗ Note

The terminal is the most powerful tool

The terminal — also known as the **command-line interface** or **shell** — is one of the most powerful and flexible tools available on a computer, especially in scientific and research computing.

At its core, the shell is a program that provides a **text-based interface** between the user and the operating system. Rather than clicking through menus and windows, users type commands directly to control the computer. This is why it is called a **"shell"** — it wraps around the underlying system and gives users a structured way to interact with it.

There are different kinds of shells (like `bash`, `zsh`, or `fish`), but they all allow the user to execute programs, navigate the filesystem, automate tasks, and combine tools in highly efficient ways.

The terminal is particularly favored by expert users and researchers because it offers:

- **Precision**: Commands can be written and saved exactly, removing ambiguity and allowing for exact replication of tasks.
- **Speed**: Complex or repetitive tasks that would take many clicks in a graphical interface can be done in seconds with a single command or script.
- **Automation**: Workflows can be scripted, making it easy to repeat analyses, preprocess data, or set up entire environments automatically.
- **Remote access**: Shell-based tools like SSH make it possible to work on powerful remote systems or supercomputers as if they were local.
- **Reproducibility**: Everything you do in the terminal can be logged and version-controlled, which is essential for transparent scientific research.

In short, while it may seem intimidating at first, the shell becomes an essential ally for researchers dealing with data, software environments, or computing infrastructure. Learning to use it effectively can significantly increase both productivity and control over computational tasks.

Consider taking the Shell Crash Course

Bonus insight for support teams

Why this matters for your role:

- Researchers will ask you **why they can't install a tool** or **access a folder** — this often comes down to OS permissions.
- In secure environments (e.g., working with personal data), understanding **user roles** and **access controls** is crucial for compliance.
- If you support computational research, you'll often interact with **Linux-based systems** and need to understand how users and files are managed there.

Step 4: Install the Tools You Are Planning to Use

Now that our operating system is running, we need to give it **specific abilities**. A base system can open windows and manage files — but it can't do research yet.

To make the computer useful for science, we need to **install software**.

What Is Software Installation?

Installing software means placing a program and its dependencies on your system so it can be run:

- It may involve downloading files,
- Unpacking or compiling them,
- Configuring settings, and
- Linking them to your operating system.

Examples:

- A climate scientist installs **NetCDF tools** to read weather models.
- A data scientist installs **R and the tidyverse** for statistical analysis.
- A linguist installs **NLTK in Python** for natural language processing.

Package Managers and Installers

Operating systems often come with tools to simplify installation:

- **Linux**:
  - `apt`, `dnf`, `yum`, `conda`, `pip`
- **Windows**:
  - `.exe` installers, Windows Store, Chocolatey
- **macOS**:
  - `.dmg` files, `brew`

Some tools are graphical (click to install), others use the **command line** (important in HPC environments and important for reproducibility).

Permissions: Can Everyone Install Software?

No. In shared or secure systems:

- Only **admins (root)** can install system-wide software.
- Regular users can only install in their **home directory** or via isolated environments.

In research groups or HPC systems, this often means:

- You ask the IT team to install something, **or**

- You use workarounds like **conda environments**, **virtualenv**, or **containers**.

Licenses and Access

Some tools are **open source** — free to use, inspect, and share:

- Python, R, Julia
- QGIS, LibreOffice, Jupyter

Others are **proprietary** or require institutional licenses:

- MATLAB, SPSS, ArcGIS, Microsoft Office

You may have to:

- Register with a license server,
- Use your university VPN,
- Or request access through IT.

Note

Containers: Software in a Box

When you can't install tools normally — or want full control over the environment — you can use **containers**:

- Tools like **Docker** or **Singularity/Apptainer** let you run pre-packaged environments.
- Everything the program needs is inside the container — like a **portable mini-kitchen**.

Use case: A data scientist runs a reproducible analysis inside a Docker container that works the same way on a laptop, cloud server, or supercomputer node.

Discussion

Cooking Metaphor: Installing Tools = Equipping Your Kitchen

- Installing **software** is like choosing your appliances and ingredients.
- Package managers are like a **kitchen supply catalog**.
- Permissions are like **access rules** in a shared kitchen — maybe only the head chef can bring in new knives.
- Containers are **your own mobile kitchen** — everything pre-installed, works the same anywhere.

Bonus Insight for support teams

Why this matters for your support role:

- Researchers may ask for help installing domain-specific software — or report that "something isn't working" on a cluster or shared VM.
- You should understand:
  - Why permission errors happen,
  - What alternatives (like `conda` or containers) are available,
  - When to escalate to IT or request a license.

You can help by maintaining a list of pre-approved tools, recommended packages, or links to shared containers.

Step 5: Expand the Computer with Remote Storage

Our computer is now running, and we've installed some tools. But soon, we'll run into a bottleneck: **storage space and speed**.

Research data can grow quickly — especially in fields like genomics, imaging, or environmental monitoring. So we need to expand beyond local disks by using **remote storage**.

Why Is Local Storage Sometimes Not Enough?

Local storage (like an SSD in your laptop or server) is:

- Fast (especially SSDs),
- Always available (no internet needed),
- Limited in size (256 GB to a few TB),
- Vulnerable to failure or theft.

Many datasets in research **exceed local capacity**:

- A single fMRI scan might be 2GB… scale that to 100 subjects and you have 200G… consider that processing requires 5 times the raw MRI data space and we have about 1TB…
- A whole-genome dataset can be over 200 GB,
- Satellite images can reach **terabytes per day**.

What Is Remote Storage?

Remote storage refers to **data stored on another machine** that your computer can access over a network. This includes:

- **Network drives** (e.g., NFS, SMB),
- **Institutional data services** (e.g., CSC Allas object storage),
- **Cloud buckets** (e.g., Amazon S3, Azure Blob),
- **Remote repositories** (e.g., Zenodo, Figshare, institutional archives).

Remote storage can be:

- Much **larger** than local disks,
- **Backed up** by IT,
- **Shared** with collaborators,
- But often **slower** to read/write due to network speed.

The I/O Bottleneck

I/O = Input/Output: how fast data is read from or written to disk.

- **Local SSDs**: Extremely fast (ideal for GPU processing, simulations, etc.).
- **Remote storage**: Depends on internet speed, protocols, and server load. Often slower and less predictable.

If you're doing **GPU-intensive computation**, reading data too slowly can cause the GPU to sit idle — wasting expensive compute time.

> **Note**
>
> **Cooking Metaphor: Storage as the Pantry**
>
> - Your **local disk** is the pantry in your own kitchen — fast, private, limited.
> - Remote storage is like a **warehouse down the street** — it's bigger, maybe shared, but slower to access.
> - For some tasks, you prep everything from your local pantry.
> - For others, you send someone (like `rsync` or a script) to fetch ingredients in bulk.
>
> Choosing where to store your data affects how fast and efficiently your recipe (research) runs.

Security and Remote Storage

Remote storage also brings **data protection considerations**:

- Is the storage **encrypted**?
- Who has **access** to it?
- Is it located in a country with appropriate **legal protections** (important for GDPR)?
- Does it offer **versioning** or **audit logs**?

Trusted research infrastructures (like CSC, SURF, or institutional servers) are often required for **personal or sensitive data**.

> **Note**
>
> **Classification of information and data storage**

Many organizations — including universities, research institutes, and government agencies — use a **classification framework** to manage information securely and responsibly. This framework helps determine **where and how different types of data should be stored, accessed, and shared**.

A widely used (though not universal) classification model includes four categories:

1. **Public**
   Information that is intended for open sharing and has no access restrictions. Examples include published research articles, public websites, and open data. This kind of information can be stored on public servers, shared via email, and posted online without special safeguards.
2. **Internal**
   Information meant for use within the organization but not intended for public release. It may include internal documentation, draft manuscripts, or general staff communications. Internal data should be stored on organizational platforms (e.g., secure shared drives or institutional cloud services), but doesn't require encryption or special permissions.
3. **Confidential**
   Sensitive information that, if exposed, could cause harm to individuals, the institution, or research partners. This includes personal data (e.g., student records, interview transcripts), research data under embargo, or licensed datasets. Confidential data **must be stored in secure environments** — such as encrypted drives, secure servers, or designated trusted research infrastructures — and access must be limited to authorized individuals only.
4. **Secret**
   Highly restricted information that could result in serious harm if disclosed. It may apply to secondary use of medical data, national security data, defense-related research, or sensitive proprietary data under strict non-disclosure agreements. Secret data typically requires isolated, access-controlled environments with strong encryption and strict auditing.

While this four-level model is common, **each organization may have its own specific definitions, labels, and handling rules** — often aligned with legal or regulatory frameworks (such as GDPR for personal data).

It is essential to **check your institution's data classification policy** and follow their guidance when deciding:

- Where to store different types of data,
- Who is allowed to access it,
- Whether encryption or special access controls are needed,
- And how data should be shared or archived.

Bonus Insight for support teams

Support teams are often responsible for:

- Advising on **where data should live** (local vs. remote),
- Managing **access permissions and documentation**,
- Explaining **performance trade-offs** between storage types,
- Ensuring compliance with policies and data management plans (DMPs).

Understanding remote storage — both **practically** and **strategically** — is key to supporting researchers.

Step 6: Expand the Computing by Connecting with Other Computers

Sometimes, our own computer — even with extra storage — **just isn't enough** to do what we need. Maybe we're processing too much data, training a large AI model, or need more memory than we physically have.

In these cases, we don't just expand storage — we expand **computation** by connecting to other computers.

Using Remote Computers

Remote computing means using another machine, typically more powerful than yours, to:

- Run programs,
- Store data,
- Train models,
- Or host services (like dashboards or APIs).

This remote computer could be:

- A **cloud-based virtual machine** (e.g. from CSC, AWS, Azure),
- A **remote workstation** in your institution,
- A **computing node** on a cluster or HPC system,
- Or a **server** that you access over the internet.

You control the remote computer using tools like:

- **SSH** (secure shell, text-based),
- **Remote desktop** (GUI-based),
- **Web interfaces** (e.g. JupyterHub, VS Code in the browser, OpenOnDemand https://www.puhti.csc.fi/public/).

Offloading Computation: APIs

In many modern applications, **your computer is only a front-end**, while the work is done elsewhere.

This is exactly how **APIs (Application Programming Interfaces)** work:

- You send a **request** (e.g. a prompt to ChatGPT),
- The remote system does all the processing,
- You get back a **response** (e.g. a completed sentence).

Other examples:

- Image classification via a web API,
- Weather forecast data fetched from a public government API,
- Research datasets accessed through SPARQL endpoints or RESTful APIs.

Your laptop just handles the input/output — **the thinking happens remotely**.

Note

An example on the terminal to query a remote API

In research, it's often useful to retrieve data from external services using APIs (Application Programming Interfaces). Many APIs are **RESTful**, meaning they use standard web protocols (like HTTP) and return data in structured formats, such as JSON. You can query these APIs directly from the terminal using a tool like `curl`.

An example with the NASA Astronomy Picture of the Day (APOD) service.

This API provides information and metadata about astronomy images curated by NASA.

You can retrieve the latest entry like this:

```
curl "https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY"
```

This command sends a GET request to the NASA APOD API using a public test key (`DEMO_KEY`). The response will be in JSON format and includes the image URL, title, date, and explanation:

```
{
  "date": "2025-04-05",
  "title": "Solar Eclipse from the ISS",
  "url": "https://apod.nasa.gov/apod/image/2504/eclipse_iss.jpg",
  "explanation": "This image from the International Space Station shows..."
}
```

You can save the response to a file for later use:

```
curl "https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY" -o apod.json
```

While this example uses a public endpoint, many APIs require authentication via API keys, tokens, or other headers — these can be added with curl options (e.g. `-H "Authorization: Bearer TOKEN"`), which we'll explore later.

Accessing APIs from the terminal allows researchers to:

- Automate data collection,
- Build reproducible workflows,
- Integrate external scientific datasets directly into analysis pipelines.

💬 Discussion

Cooking Metaphor: Borrowing Another Kitchen

Imagine your kitchen is too small for a banquet. You can:

- Rent space at a **larger kitchen across town** (remote machine),
- Hire a **chef robot** that cooks on command (API),
- Or plug into a **remote food factory** using only your computer as a control panel.

Your local computer becomes a **terminal** — a keyboard and a screen — while the real work happens remotely.

Cloud Computing and Virtual Machines

Cloud providers offer **on-demand computing environments**:

- You create a virtual machine (VM) with a few clicks,
- Choose the number of CPUs, RAM, and even GPUs,
- Install your tools, run your code, and shut it down when done.

Cloud platforms include:

- **General-purpose**: AWS, Google Cloud, Azure
- **Research-focused**: CSC

The upside: scale, flexibility, reproducibility.
The downside: complexity, cost, and security considerations.

Security and Compliance

Using remote systems for research means:

- Being mindful of **who else can access the system**,
- Ensuring **data protection** (especially with personal data),
- Using **strong authentication** (SSH keys, 2FA),
- Following your institution's **IT and ethical guidelines**.

Bonus Insight for support teams

As a support person, you'll often help researchers:

- Understand where to run their code or models,
- Set up SSH access or Jupyter environments,
- Decide between **on-premises** and **cloud-based** resources,
- Comply with **data security and legal requirements** when using external computing power.

You don't need to be a system administrator — but you *do* need to know what's possible, what's allowed, and what questions to ask.

Step 7: Connect Multiple Computers Together to Build the Supercomputer

Now that you understand how to use one remote computer, it's time to **scale up**.

A **supercomputer** isn't one big machine — it's a **network of many smaller computers (called nodes)** connected with high-speed communication links and managed by special software to behave like one enormous machine.

What Is a Supercomputer Made Of?

A typical supercomputer consists of:

- **Compute nodes**: Each node is like a standalone computer with CPUs, RAM, and sometimes GPUs.
- **Interconnects**: Fast network cables (e.g., InfiniBand) that let the nodes talk to each other with minimal delay.
- **Storage nodes**: Shared data storage for all the nodes to read/write from.
- **Login nodes**: Where users connect to prepare their jobs.
- **Management system**: To monitor hardware, schedule jobs, and manage user activity.

A supercomputer can include **thousands of nodes**, working together.

How Does It Work in Practice?

Researchers **submit jobs** to a queue. These jobs might:

- Simulate complex phenomena (e.g. earthquakes, airflow),
- Process thousands of images or genome files,

- Train huge machine learning models.

A **job scheduler** (like Slurm, PBS, or Torque) decides:

- When the job runs,
- Which nodes to assign,
- How to balance workloads fairly.

Jobs can run in parallel across dozens or hundreds of nodes — **a form of teamwork at digital scale.**

High-Speed Connectivity: InfiniBand

For nodes to work in parallel efficiently, they must communicate very quickly.
InfiniBand is a specialized networking technology with **very low latency and high bandwidth**, often used in supercomputers.

Unlike Wi-Fi or standard Ethernet, InfiniBand ensures that node-to-node communication doesn't become the bottleneck.

Why Supercomputers Are Shared Resources

Running and maintaining a supercomputer is expensive and energy-intensive. That's why most supercomputers:

- Are **shared national or regional infrastructure** (e.g., CSC's Mahti in Finland),
- Serve **multiple research groups** at the same time,
- Are managed by IT teams and access is granted via applications or merit.

This model is not just efficient — it's **sustainable and equitable**.

Note

Understanding parallelisation with a kitchen metaphor

If it takes 20 minutes to cook a pasta from beginning to end, having 4 stoves or even 4 entire kitchens does not mean that you can cook pasta in 20 / 4 = 5 minutes. Some part of the process can be parallelised, but the cook (= the program) needs to know what to parallelise.

See expanded kitchen metaphor here and here. There are also video version of those slides under Aalto Scientific Computing YouTube channel.

Discussion

Cooking Metaphor: A Shared Industrial Kitchen

- Each node is a **kitchen station**, equipped for specific tasks.
- They are connected by **a conveyor belt system** (InfiniBand),
- Coordinated by a **head chef (Slurm)** who decides who cooks what and when.
- Multiple cooking teams share the kitchen, each getting a time slot.

Instead of every research group building their own kitchen, they **share the industrial space** — everyone gets better tools, and no one wastes energy duplicating them.

Bonus Insight for support teams

Your role could involve:

- Helping researchers access HPC resources (e.g., apply for accounts, get computing quotas),
- Explaining how job scheduling works,
- Clarifying storage policies and best practices,
- Assisting with reproducibility (e.g., using containers or version control in HPC pipelines),
- Navigating the **research governance** around using shared infrastructure.

Supercomputers are powerful — but without good planning and support, they can be hard to use well.

Step 8: Celebrate! You Now Have Built a Supercomputer!

Congratulations — you've just walked through the conceptual journey of building a supercomputer!

You started with:

- A single computer and its parts,
- Brought it to life with BIOS and an operating system,
- Installed tools,
- Expanded with remote storage and remote computing,
- And finally, **connected many computers into a coordinated, powerful infrastructure**.

You now understand the **fundamentals of computational research infrastructure** — and the role it plays in modern science.

What Comes Next?

While building the hardware is important, **building understanding is even more powerful**. As a researcher or research supporter, your next steps could include:

Learn how to *use* supercomputers:

- Connect to login nodes with `ssh`,
- Submit jobs with Slurm,

- Use modules to load software,
- Explore user guides from national computing centers (e.g., CSC, SURF, NeIC).

Learn about reproducible environments:

- Use containers (Docker, Singularity),
- Practice setting up workflows with Snakemake or Nextflow,
- Document tools and versions for long-term reproducibility.

Learn about responsible use:

- Understand the legal and ethical frameworks (e.g., GDPR, data classification),
- Know when personal or sensitive data **can or cannot** be processed remotely,
- Get familiar with data access policies, security settings, and logging tools.

Support researchers:

- Help them write better data management plans,
- Translate their research needs into technical requirements,
- Connect them with IT and infrastructure services.

💬 Discussion

You've built your own kitchen, learned to borrow others, and now you're part of a shared cooking facility where world-class science happens.
You're not just a user of tools — you help **design kitchens**, **organize workflows**, and **support researchers** in creating sustainable, powerful recipes for discovery.

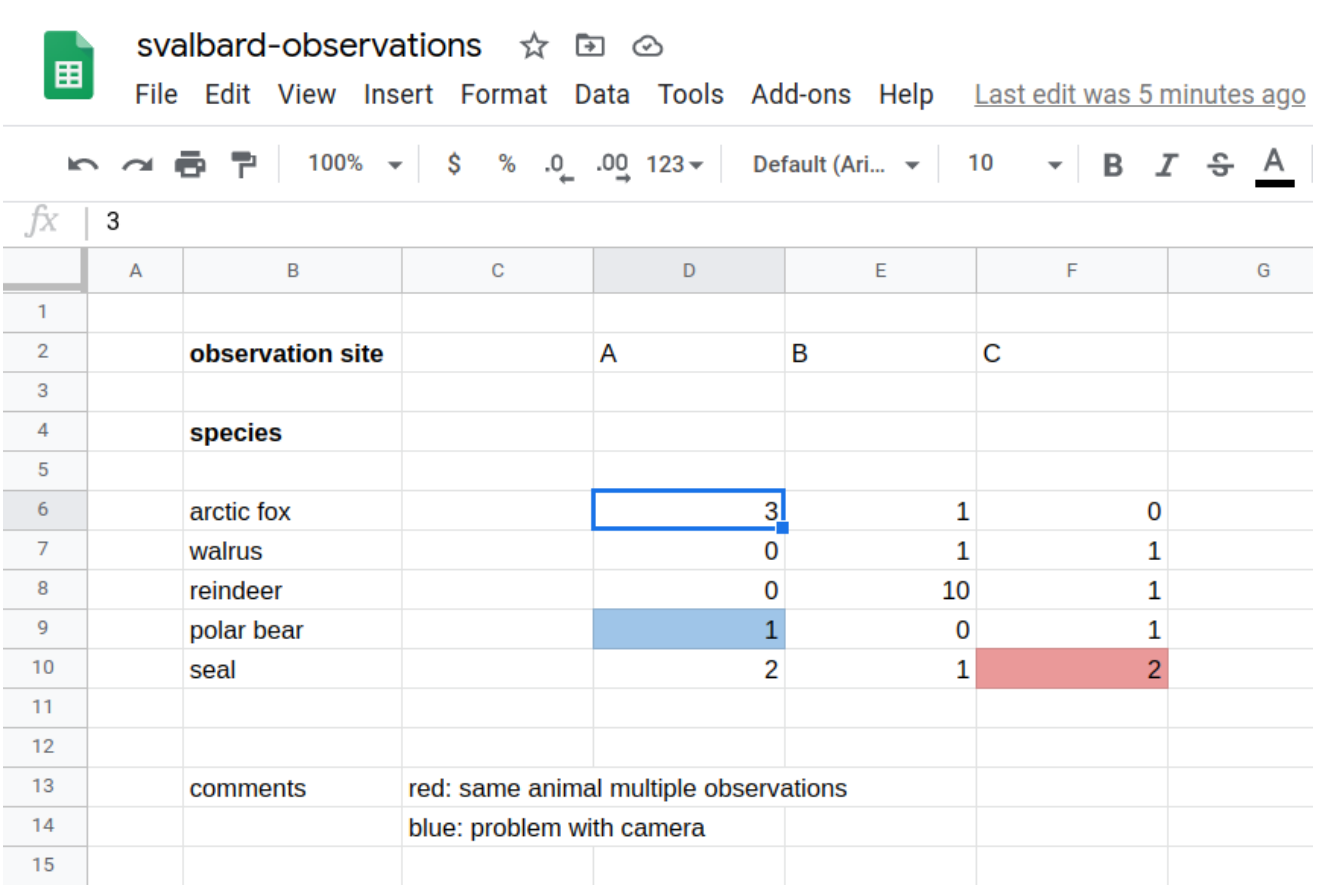
Remember: you are never alone

High-performance computing is a diverse community of people with various expertise who are very happy to discuss research data analysis workflows and decide together which tool is the most suitable for which case. It can be useful to also learn "how to ask for help with (super)computers".

Tidy data and dealing with messy data

❗ Objectives

- Knowing about the tidy data format
- Be able to reformat tabular data into the tidy data format



	A	B	C	D	E	F	G
1							
2		**observation site**		A	B	C	
3							
4		**species**					
5							
6		arctic fox		3	1	0	
7		walrus		0	1	1	
8		reindeer		0	10	1	
9		polar bear		1	0	1	
10		seal		2	1	2	
11							
12							
13		comments	red: same animal multiple observations				
14			blue: problem with camera				
15							

Example spreadsheet (this is a phantasy dataset, apologies to biology students/researchers - this is not my domain).

💬 What is the problem with storing data like this?

- Format: Limited interoperability with other programs
- Error prone (see e.g. this famous example)
- Difficult to parse ("understand") by scripts: difficult to automate
- Not in *tidy format*: difficult to extend/modify

How should we arrange the data?

Species	Observation sites
arctic fox	A, B
walrus	B, C
reindeer	B, C
polar bear	A, C
seal	A, B, C

Attempt 1: Not great since we need to somehow divide at the comma. How should we deal with multiple sightings?

Species	Observation site A	Observation site B	Observation site C
arctic fox	3	1	0
walrus	0	1	1
reindeer	0	10	1
polar bear	1	0	1
seal	2	1	2

Attempt 2: Adding observation sites will force us to add columns.

Species	arctic fox	walrus	reindeer	polar bear	seal
Observation site A	3	0	0	1	2
Observation site B	1	1	10	0	1
Observation site C	0	1	1	1	2

Attempt 3: Adding species will force us to add columns.

Species	Observation site	Number of sightings
arctic fox	A	3
arctic fox	B	1
walrus	B	1
walrus	C	1
reindeer	B	10
reindeer	C	1
polar bear	A	1
polar bear	C	1
seal	A	2
seal	B	1
seal	C	2

Tidy data format: Columns are variables, rows are observations/measurements. Easy to add new species and sites.

Tidy data format

- Hadley Wickham: Tidy Data
- Columns are variables
- Rows are observations/measurements
- “Long form”
- Order does not matter
- **Easy to extend** with more species and more sites without modifying the code
- **Structure for storing data** - this does not mean that this is ideal for tables in presentations or publications
- It is possible to convert between wide form and long form and back (e.g. using `pandas.melt` or `pandas.pivot`), see this example notebook

Use a standard format

```
Species,Observation site,Number of sightings
arctic fox,A,3
arctic fox,B,1
walrus,B,1
walrus,C,1
reindeer,B,10
reindeer,C,1
polar bear,A,1
polar bear,C,1
seal,A,2
seal,B,1
seal,C,2
```

- **Use a format that is standard in your community, don't invent your own**
- CSV is often a good choice since most visualization tools can read CSV data

There are many more formats (adapted after Python for Scientific Computing):

Name:	Human readable:	Space efficiency:	Arbitrary data:	Tidy data:	Array data:	Long term storage/sharing:
CSV	✅	❌	❌	✅	🟨	✅
Feather	❌	✅	❌	✅	❌	❌
Parquet	❌	✅	🟨	✅	🟨	✅
NPY	❌	🟨	❌	❌	✅	❌
HDF5	❌	✅	❌	❌	✅	✅
NetCDF	❌	✅	❌	❌	✅	✅
JSON	✅	❌	🟨	❌	❌	✅
GeoJSON	✅	❌	🟨	❌	❌	✅
Excel	❌	❌	❌	🟨	❌	🟨
Graph formats	🟨	🟨	❌	❌	❌	✅
SQL	❌	🟨	❌	❌	❌	❌

Note

- ✅ : Good
- 🟨 : Ok / depends on a case
- ❌ : Bad

✍ Exercise

We have been given a dataset so that for a given book we know how many copies we have stored in the main public libraries in Helsinki

Book Title	Pasila Library	Oodi Library	Kallio Library
Risto Räppääjä ja kauhea makkara	2	1	3
Heinähattu ja Vilttitossu	1	0	2
Tatu ja Patu Helsingissä	3	2	1
Risto Räppääjä saa isän	2	2	0
Herra Hakkarainen maailmalla	1	3	2
Koiramäen talossa	0	2	1
Ella ja kaverit salaisessa palveluksessa	3	1	2

- Is the data **tidy**?
- What is the tidy version of that data?

✔ Solution

Book Title	Library	Copies
Risto Räppääjä ja kauhea makkara	Pasila Library	2
Risto Räppääjä ja kauhea makkara	Oodi Library	1
Risto Räppääjä ja kauhea makkara	Kallio Library	3
Heinähattu ja Vilttitossu	Pasila Library	1
Heinähattu ja Vilttitossu	Oodi Library	0

Book Title	Library	Copies
Heinähattu ja Vilttitossu	Kallio Library	2
Tatu ja Patu Helsingissä	Pasila Library	3
Tatu ja Patu Helsingissä	Oodi Library	2
Tatu ja Patu Helsingissä	Kallio Library	1
Risto Räppääjä saa isän	Pasila Library	2
Risto Räppääjä saa isän	Oodi Library	2
Risto Räppääjä saa isän	Kallio Library	0
Herra Hakkarainen maailmalla	Pasila Library	1
Herra Hakkarainen maailmalla	Oodi Library	3
Herra Hakkarainen maailmalla	Kallio Library	2
Koiramäen talossa	Pasila Library	0
Koiramäen talossa	Oodi Library	2
Koiramäen talossa	Kallio Library	1
Ella ja kaverit salaisessa palveluksessa	Pasila Library	3
Ella ja kaverit salaisessa palveluksessa	Oodi Library	1
Ella ja kaverit salaisessa palveluksessa	Kallio Library	2

Reference

Some reference page

Credit

When preparing this lesson, we have reused these resources:

- https://coderefinery.github.io/data-visualization-python/
- https://aaltoscicomp.github.io/python-for-scicomp/