Data Handling: Import, Cleaning and Visualisation

Lecture 3:
A Brief Introduction to Data and Data Processing

Prof. Dr. Ulrich Matter, updated by Dr. Aurélien Sallin

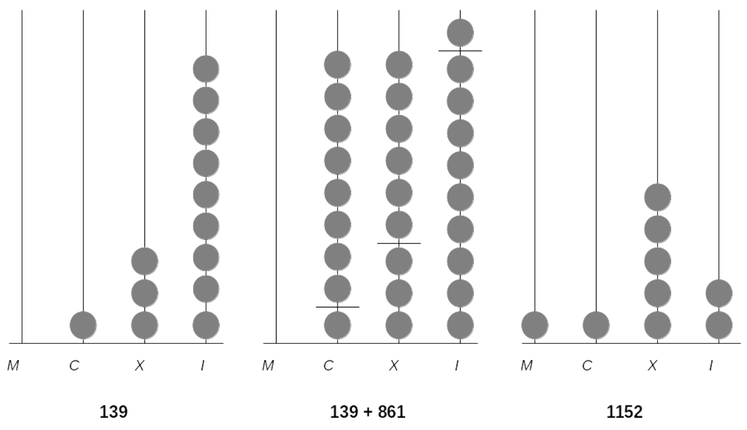
Processing Data

Simple calculations in numeral systems

How does the abacus work?

- Need to agree on a standard (numeral system)!
- Define the *base* of the frame's numeral system.
- For example, a Roman abacus with base 10.

The Roman abacus (base 10)



The Roman abacus (base 10)

The example in more detail:

- Columns reflect the positions of the digits
- and signify the power of 10 with which the digit is multiplied:

$$139 = (1 \times 10^2) + (3 \times 10^1) + (9 \times 10^0)$$

- *Base 10*: 10 different signs (0-9) to distinguish one-digit numbers.

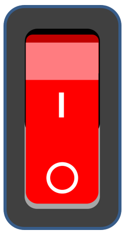
Base 10: the 'decimal system'

- What we work with in everyday life.
- What we encounter in undergraduate math, statistics, etc.
- However, not the relevant base once we use electronic tools.

The binary system

Microprocessors can only represent two signs (states):

- 'Off' = `0`
- 'On' = `1`



The binary counting frame

- Only two signs: `0` , `1` .
- Base 2.
- Columns: $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, and so forth.

The binary counting frame

- Sufficient to represent all *natural* numbers in the decimal system.
- Representing fractions is tricky
  - e.g. $1/3 = 0.333..$ actually constitutes an infinite sequence of 0s and 1s.
  - Solution: 'floating point numbers' (not 100% accurate)

The binary counting frame

What is the decimal number *139* in the binary counting frame?

- Solution:

$$(1 \times 2^7) + (1 \times 2^3) + (1 \times 2^1) + (1 \times 2^0) = 139.$$

- More precisely:

$$\begin{aligned}(1 \times 2^7) + (0 \times 2^6) + (0 \times 2^5) + (0 \times 2^4) + (1 \times 2^3) \\ + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) = 139.\end{aligned}$$

- That is, the number `139` in the decimal system corresponds to `10001011` in the binary system.

Decimal numbers in a computer

If computers only understand `0` and `1` , how can they express decimal numbers like *139*?

- *Standards* define how symbols, colors, etc are shown on the screen.
- Facilitates interaction with a computer (our keyboards do not only consist of a `0` / `1` switch).

Express decimal numbers in binary values

- *How to translate from one to the other?*

- Draw a “binary abacus” in the form of a table.
- Fill in 0s and 1s in the corresponding columns to “select” and sum up the 2^i.

Number	128	64	32	16	8	4	2	1
0 =	0	0	0	0	0	0	0	0
1 =	0	0	0	0	0	0	0	1
2 =	0	0	0	0	0	0	1	0
3 =	0	0	0	0	0	0	1	1
…								
139 =	1	0	0	0	1	0	1	1

The hexadecimal system

- Binary numbers can become quite long rather quickly.
- Computer Science: refer to binary numbers with the *hexadecimal* system.

The hexadecimal system

- *16 symbols*:
  - `0` - `9` (used like in the decimal system)…
  - and `A` - `F` (for the numbers 10 to 15).
- *16 symbols* >>> *base 16*: each digit represents an increasing power of 16 (16^0, 16^1, etc.).

The hexadecimal system

What is the decimal number 139 expressed in the hexadecimal system?

- Solution:

$$(8 \times 16^1) + (11 \times 16^0) = 139.$$

- More precisely:

$$(8 \times 16^1) + (B \times 16^0) = 8B = 139.$$

- Hence: `10001011` (in binary) = `8B` (in hexadecimal) = `139` in decimal.

The hexadecimal system

Advantages (when working with binary numbers)

1. Shorter than raw binary representation
2. Much easier to translate forth and back between binary and hexadecimal than binary and decimal.

WHY?

- Hexadecimal digits can always be represented in four-digit binary equivalent: because it is in base 16, each sincle digit in hexadecimal can be represented by 4 digints in binary.
- $8 = 1000$, $B = 11 = 1011$, thus…
- `8B` (in hexadecimal) = `10001011` (`1000 1011`) in binary.

Character Encoding

Computers and text

How can a computer understand text if it only understands `0` s and `1` s?

- *Standards* define how `0` s and `1` s correspond to specific letters/characters of different human languages.
- These standards are usually called *character encodings*.
- Coded character sets that map unique numbers (in the end in binary coded values) to each character in the set.
- For example, ASCII (American Standard Code for Information Interchange) or utf-8.

ASCII logo. (public domain).

ASCII Table (examples)

Binary	Hexadecimal	Decimal	Character
0011 1111	3F	63	`?`
0100 0001	41	65	`A`
0110 0010	62	98	`b`

Character encodings: why should we care?

- In practice, Data Science means handling digital data of all formats and shapes.
  - Diverse sources.
  - Different standards.
  - *read*/*store* data.
- At the lowest level, this means understanding/handling encodings.

Computer Code and Text-Files

Putting the pieces together…

Two core themes of this course:

1. How can *data* be *stored* digitally and be *read* by/imported to a computer?
2. How can we give instructions to a computer by writing *computer code*?

In both of these domains we mainly work with one simple type of document: *text files*.

Text-files

- A *collection of characters* stored in a designated part of the computer memory/hard drive.
- An easy to read representation of the underlying information (`0` s and `1` s)!
- Common device to store data:
  - Structured data (tables)
  - Semi-structured data (websites)
  - Unstructured data (plain text)
- Typical device to store computer code.

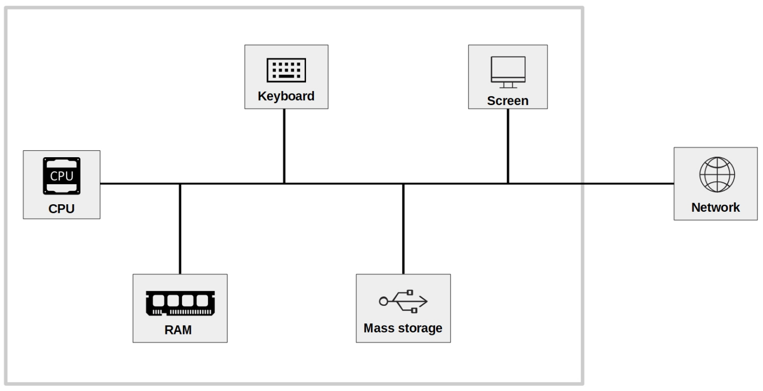
Text-editors: RStudio, Atom

Text editors are software tools that help you edit text-files for the purpose of coding/programming and/or for the purpose of storing/editing data in an accessible format.

- To write a computer program, all you need is a text editor and the interpreter (or compiler), which translates the code (in our case in the R language) into machine code (instructions in 0s and 1s).
- To store data in a simple but meaningful way on a computer, all you need is a text editor and a standard/schema which gives the data records some structure.

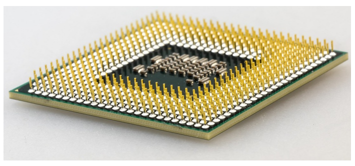
Data Processing Basics

Components of a standard computing environment



Basic components of a standard computing environment.

Central Processing Unit (CPU)



The "heart" of the computer. Does the computation. Processes the machine code instructions.

Random Access Memory (RAM)

Holds all the currently needed environments, objects, etc. Provides the currently processed data to the CPU. Is filled, once you start your computer, is emptied once you switch off your computer. Standard data processing techniques rely on the dataset (and more) fitting into RAM (can be an issue when working with large datasets).

Mass storage: hard drive

This is where the data (and operating system, software, etc.) is stored in the long run. When you boot your computer, the core programs needed to run the computer are loaded from the hard drive into RAM. When you *import* a dataset into R/RStudio, the data is read from a (text-)file stored on the hard drive into RAM.

Simple example of difference between RAM and hard drive: When you open a new R-Script (or Word-file, etc.), type something in it, this means you write something to an object located somewhere in RAM. If you close the application without saving the file, it will be lost. Once you “save” the script/document, it will be written to the hard drive.

Network: Internet, cloud, etc.

With today’s abundance of digital data on all kinds of economic activities, many economic research projects rely on either gathering the raw data from web sources and/or on storing the data for analytics purposes on a database or simple storage service in the web/in the cloud. In practice, datasets for analytics purposes are thus often transferred between a web source and the local hard drive, or directly read from a web source into RAM.

Putting the pieces together…

Recall the initial example (survey) of this course.

1. Access a website (over the Internet), use keyboard to enter data into a website (a Google sheet in that case). That is, data is typed into RAM and then transferred over a network connection to a web server, and stored on its hard drive.
2. An R program accesses the data of the Google sheet (again over the Internet), downloads the data, and loads it into RAM.
3. Data processing: an R program is run through the CPU to produce output (in the form of statistics/plots), the output is shown on screen.