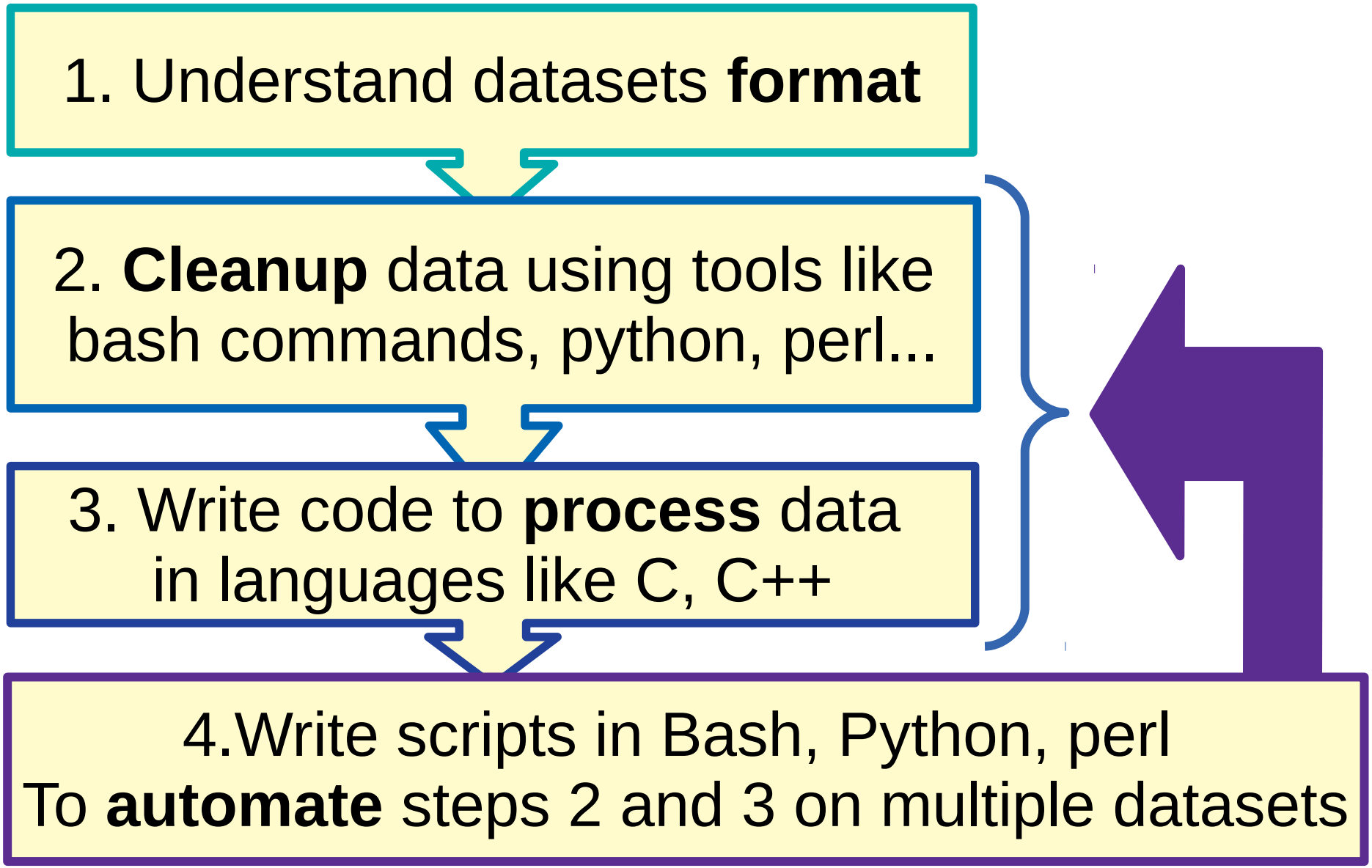
Handling datasets

Florido Paganelli
Lund University
florido.paganelli@hep.lu.se
Fysikum, Hus A, Room 403

Support:
- send me an email or use Canvas
- personal Zoom room:
https://lu-se.zoom.us/j/2485752983

Typical scientist workflow summary



Typical scientist workflow

- **Someone** (usually your supervisor, today is me) gives you reference to some data and some obscure code written by elders who now moved to the end of the known universe
  - Nobody knows what the **data** looks like and what it contains – just that is it about your science!
  - Nobody has any idea what the **code** is like and how to change it. No documentation, no one left alive to tell you
- **You** have to figure out most of the the details by yourself
- In these slides a few tips to survive

Datasets

- A dataset is some digital collection, maybe a file or a set of files, that **contains data** we want to use. Here are the typical traits of a dataset:
- **Scope**: The knowledge area it targets. Examples: archeology, budget, weather forecast...
- **Format**
  - A *format* is a **set of rules** that define in a rigorous manner how the content of the dataset should be read and written, what are their meanings and the relationship among the dataset information
  - The format can be a well know data format, more or less standardized, or some custom data format created by some community.
  - A **description** of the format is usually provided by the community that generated the dataset. It is very rare that a dataset contains information about its format.
  - Very common format **names**
    - **CSV** (**C**omma **S**eparated **V**alues)
    - **XML** (e**X**perimental **M**arkup **L**anguage)
    - **JSON** (**J**ava**S**cript **O**bject **N**otation)
  - Sometimes the format can be made explicit in the **file extension**, but it is not necessary: `fridaythe13.json`
- **Meaning of each data entry:** what is the data **exactly** about.
  - In a budget, it's probably whatever concerned an expense, the goods bought, how they were bought, maybe billing numbers...

Example from project

filename: `smhi-opendata_1_53260_20210906_214756.csv`

```
Stationsnamn;Klimatnummer;Mäthöjd (meter över marken)
Ystad;53260;2.0
Parameternamn;Beskrivning;Enhet
Lufttemperatur;momentanvärde, 1 gång/tim;degree celsius
Tidsperiod (fr.o.m);Tidsperiod (t.o.m);Höjd (meter över havet);Latitud (decimalgrader);Longitud
(decimalgrader)
1949-01-01 00:00:00;1963-09-16 23:59:59;13.0;55.4410;13.8278
1963-09-17 00:00:00;1983-08-31 23:59:59;32.0;55.4410;13.8278
Datum;Tid (UTC);Lufttemperatur;Kvalitet;;Tidsutsnitt:
1949-01-01;00:00:00;3.5;Y;;Kvalitetskontrollerade historiska data (utom de senaste 3 mån)
1949-01-01;06:00:00;3.5;Y;;Tidsperiod (fr.o.m.) = 1949-01-01 00:00:00 (UTC)
1949-01-01;12:00:00;4.2;Y;;Tidsperiod (t.o.m.) = 1983-08-31 23:59:59 (UTC)
1949-01-01;18:00:00;5.5;Y;;Samplingstid = Ej angivet
1949-01-02;00:00:00;4.3;Y;;
1949-01-02;06:00:00;4.6;Y;;Kvalitetskoderna:
1949-01-02;12:00:00;6.2;Y;;Grön (G) = Kontrollerade och godkända värden.
1949-01-02;18:00:00;5.0;Y;;Gul (Y) = Misstänkta eller aggregerade värden. Grovt kontrollerade
1949-01-03;00:00:00;4.2;Y;;
1949-01-03;06:00:00;4.0;Y;;Orsaker till saknade data:
1949-01-03;12:00:00;4.6;Y;;  stationen eller givaren har varit ur funktion.
1949-01-03;18:00:00;5.0;Y
1949-01-04;00:00:00;3.2;Y
1949-01-04;06:00:00;1.6;Y
1949-01-04;12:00:00;3.0;Y
1949-01-04;18:00:00;2.0;Y
1949-01-05;00:00:00;1.5;Y
1949-01-05;06:00:00;-0.8;Y
```

Finding clues

- **Tip 1: Search for hints**
  - look at the link, the filename, the filename **extension** (the part of the name that comes after a dot **.**)
  - Check the top of the file
  - Look for recurring keywords
  - Look how the keywords repeat, can you guess there is a structure?
    - Try to distinguish between data (values) and metadata (description of values/structure)
  - Look at the numbers/text. Can they be related to something you know, just by common sense?
- **Tip 2: Search for format details**
- Write in a search engine **“<format name> format example**”
- Formal specifications come in the form of an RFC (Request For Comments) but not easy to read.
  Search for them with the “**<format name> specification**”
- Example: “CSV specification RFC”

Sample data file: investigation

```
<?xml version="1.0" encoding="UTF-8" ?>

<Data>
<Game>
<id>1558</id>
<GameTitle>Harvest Moon Animal Parade</GameTitl
<ReleaseDate>11/10/2010</ReleaseDate>
<Platform>Nintendo Wii</Platform>
</Game>
<Game>
<id>32234</id>
<GameTitle>Busy Scissors</GameTitle>
<ReleaseDate>11/02/2010</ReleaseDate>
<Platform>Nintendo Wii</Platform>
</Game>
<Game>
<id>890</id>
<GameTitle>Rayman Raving Rabbids TV Party</Game
<ReleaseDate>11/18/2008</ReleaseDate>
<Platform>Nintendo Wii</Platform>
</Game>
<Game>
<id>908</id>
<GameTitle>Super Mario Galaxy 2</GameTitle>
<ReleaseDate>05/23/2010</ReleaseDate>
<Platform>Nintendo Wii</Platform>
</Game>
```

What can we say by **observing** this data?

- It seems to be structured in some way.
- There is some metadata information at the top that might hint at some known format. Search “XML” on google?

Can we guess something **about the structure**?

- It seems to have opening and closing tags <tag></tag>
- The tags seems to represent a tree structure

Can we guess something **about the content**? Anything you may know about?

- Clearly seems to speak about games of some kind
- Platform seems to hint to some kind of device, there is the name of some company in it

Datasets can be “dirty”

- The data is not always as you expect. Close inspection might reveal inconsistencies and corner cases that have to be “*sanitized*” or “*validated*”, or simply you need a subset of the whole dataset. In any case, something that requires special care.
- In most cases you will need to *rework the dataset* in order to process it with your code
- In any case, **never tamper the original dataset**. Do all the changes on a **separate copy**.
  - Example: take only games whose name starts with B, G or Z: create new file with such content, do not modify the original database files so that you can have them as references.
- Devil is in the detail:
  - **Encodings**. To show the content of a file, especially a text file, an operating system has to know the encoding of a file.
  - look for **invisible or non-ASCII characters.** These are usually symbols for non English-US languages, or **control characters**

What to “clean”?

- **Tip 3: Format consistency check**: check that the dataset is consistent with the format it claims/you think it is. Remove or rework any inconsistencies. Examples:
  - Additional lines/character/spaces you do not care about in your data analysis or that scramble the data format or make it hard to process via your code. For example, a csv that uses commas as separator has also commas in the data itself.
    - Always try to **resolve** the inconsistency **so to keep the data**
    - Possible solution: Carefully rewrite the relevant data or dataset with a different separator
    - If the above is not possible, **assess the data loss** caused by excluding that data: will it completely change your analysis? By how much?
    - Possible solution: on a file of 2000 lines, if 10 are bad it’s probably ok to discard them.
- **Tip 4: Meaning consistency check**: treat carefully or try to spot anything that seems like obviously wrong data, like bad reading from a sensor or a typo in the data entry Examples:
  - Suppose you have a file with temperatures from a data station. If there are temperatures like 1000 degree celsius, something is probably wrong with the data station, you should discard such data and assess how much of the datasets presents these wrong values.
  - Possible solutions:
    - on a file of 2000 lines, if 10 have strange data it’s probably ok to discard them.
    - If you’re unsure about how unrealistic the data is, perform multiple analysis **with and without** the strange data and assess what difference does it make.

Encodings

- The encoding is a **table** that maps bytes contained inside the file to a set of graphical representations. Some files carry this information at the beginning of the file, but for most text files this needs to be guessed using a *magic number.*
  Most editors can guess automatically and allow you to force-save in some encoding. In linux you can check the encoding of a file with the `file` command.
  ```
  file <filename>
  file -i <filename> # identify "mime type", used in the web
  ```
- Most common encoding sets for text files are:
  - ASCII (US-english, Latin)
  - UTF-8 (US-english and Latin with extended chars like öää)
  - UTF-16 (Symbol languages (Asian, Arabic, Hindi...))
- On most editors you can read the encoding at the bottom of the window.
- **Tip 5: encoding check**. Always check that you're reading the correct encoding.
- See CodeBlocks example in MNXB01-manual A.3.1

Control Characters

- They’re always there especially in text files. Common examples are newlines:
  - (Most linux-unix) Line Feed (**LF**): Makes a text file go to the next line. Usually represented as `\n`
  - (Mac OS) Carriage Return (**CR**): makes a text file go to the beginning of a line. Usually represented as `\r`
  - (Windows) (**CR LF**): makes a text file go to the beginning of the line and then to a new line. Usually represented in programming languages as `\r\n`
  - More info on https://en.wikipedia.org/wiki/Newline
- You can see them with `less -u <filename>` or with geany through the menu View→”Show line endings”
- The symbols \ is used to represent “escape sequences”, used for special control characters. Some other common ones:
  - \t : tab, a fixed size set of spaces
  - \s : a space
- **Tip 6: hunt for unexpected control sequences.** Always check that the line terminators are the ones you expect and that there is no weird control sequences.
- See CodeBlocks example in MNXB01-manual A.3.1

Tech advice on cleaning up a dataset

- In the case of text-file datasets, usually the best is to use tools that were created to handle text – or the so-called **string** datatype
  - **Tip 7: avoid C and C++ for working on strings.**
    C and C++ are notoriously **very bad with strings:**
    DON'T USE THEM IF POSSIBLE
  - **Tip 8: Best languages for string manipulation are perl, python and bash commands.**
    The cleanup is easier if done with any of the above languages.
- In the course project you will attempt to automate a workflow where the data needs to be cleaned up first
- I am teaching bash, but there's nothing preventing you from using any other tools you fancy such as python or Microsoft Excel or Google Spreadsheet.
  The use if bash is just practical if you have hundreds of text files to parse, and all the tools are for free.
  - You can run an ubuntu/linux-like shell also on windows (e.g. cygwin) without virtualization. The use of these tools will become more and more common, see the MNXB01-manual about running your own linux for details!

Workflows with various tools

- In the free software/open source community everyone shares knowledge about coding.
  - This usually means that someone's work is based on someone else's
  - This generated a style of creating software that is a mix of different programming languages, tools, practices:
    composing different applications to achieve a goal
- It is very common that your C++ code will require some **preparation** before the build for which leads to **tedious repetitive** commands to type in
- **"tedious repetitive"** is what a computer is good at.
  Tip 9: use a computer to do tedious and repetitive stuff
  A human has better things to do in life than monkey-coding!
- *Scripting languages* are very good to automate boring work.

A case study

- We want to go from the file format on the left side to the one on the right side.

```
Stationsnamn;Klimatnummer;Mäthöjd (meter över marken)
Ystad;53260;2.0

Parameternamn;Beskrivning;Enhet
Lufttemperatur;momentanvärde, 1 gång/tim;degree celsius

Tidsperiod (fr.o.m);Tidsperiod (t.o.m);Höjd (meter över havet);Latitud (decimalgrader);Longitud
(decimalgrader)
1949-01-01 00:00:00;1963-09-16 23:59:59;13.0;55.4410;13.8278
1963-09-17 00:00:00;1983-08-31 23:59:59;32.0;55.4410;13.8278

Datum;Tid (UTC);Lufttemperatur;Kvalitet;;Tidsutsnitt:
1949-01-01;00:00:00;3.5;Y;;Kvalitetskontrollerade historiska data (utom de senaste 3 mån)
1949-01-01;06:00:00;3.5;Y;;Tidsperiod (fr.o.m.) = 1949-01-01 00:00:00 (UTC)
1949-01-01;12:00:00;4.2;Y;;Tidsperiod (t.o.m.) = 1983-08-31 23:59:59 (UTC)
1949-01-01;18:00:00;5.5;Y;;Samplingstid = Ej angivet
1949-01-02;00:00:00;4.3;Y;;
1949-01-02;06:00:00;4.6;Y;;Kvalitetskoderna:
1949-01-02;12:00:00;6.2;Y;;Grön (G) = Kontrollerade och godkända värden.
1949-01-02;18:00:00;5.0;Y;;Gul (Y) = Misstänkta eller aggregerade värden. Grovt kontrollerade
1949-01-03;00:00:00;4.2;Y;;
1949-01-03;06:00:00;4.0;Y;;Orsaker till saknade data:
1949-01-03;12:00:00;4.6;Y;;  stationen eller givaren har varit ur funktion.
1949-01-03;18:00:00;5.0;Y
1949-01-04;00:00:00;3.2;Y
1949-01-04;06:00:00;1.6;Y
1949-01-04;12:00:00;3.0;Y
1949-01-04;18:00:00;2.0;Y
1949-01-05;00:00:00;1.5;Y
1949-01-05;06:00:00;-0.8;Y
```

Not very easy to process:
- Metadata headers
- inconsistent field structure, some fields are used for comments
- semicolon may not be nice for C++

```
1949-01-01 00:00:00 3.5 Y
1949-01-01 06:00:00 3.5 Y
1949-01-01 12:00:00 4.2 Y
1949-01-01 18:00:00 5.5 Y
1949-01-02 00:00:00 4.3 Y
1949-01-02 06:00:00 4.6 Y
1949-01-02 12:00:00 6.2 Y
1949-01-02 18:00:00 5.0 Y
1949-01-03 00:00:00 4.2 Y
1949-01-03 06:00:00 4.0 Y
1949-01-03 12:00:00 4.6 Y
1949-01-03 18:00:00 5.0 Y
1949-01-04 00:00:00 3.2 Y
1949-01-04 06:00:00 1.6 Y
1949-01-04 12:00:00 3.0 Y
1949-01-04 18:00:00 2.0 Y
1949-01-05 00:00:00 1.5 Y
1949-01-05 06:00:00 -0.8 Y
```

Simplified raw data
- No headers, just raw data
- Removal of unused fields
- spaces instead of semicolon

- See the example code at https://github.com/floridop/MNXB01-2021/blob/main/floridopag/tutorial3/casestudy/code/smhicleaner.sh.solution

Dataset Quiz

- Click these links:
  - https://github.com/floridop/MNXB01-2020/blob/master/floridopag/lecture2/lecture2examples/data/nintendowiigames.xml
  - http://musicbrainz.org/ws/2/artist/5b11f4ce-a62d-471e-81fc-a69a8278c7da?inc=aliases&fmt=json
  - http://www.smhi.se/pd/klimat/ozone/data//oz2019.vin
- For each of the above:
  - What is it about (scope)?
  - What is the format?
  - What is the actual information contained?

Dataset quiz - solutions

- Regarding these links:
  a) https://github.com/floridop/MNXB01-2020/blob/master/floridopag/lecture2/lecture2examples/data/nintendowiigames.xml
  b) http://musicbrainz.org/ws/2/artist/5b11f4ce-a62d-471e-81fc-a69a8278c7da?inc=aliases&fmt=json
  c) http://www.smhi.se/pd/klimat/ozone/data//oz2019.vin

1) Could you guess what each dataset about?
   a) Nintendo WII videogames (check the `Platform` tag or the filename)
   b) Music metadata (check the main website musicbrainz.org)
   c) Check the URL, it's the Swedish weather forecast services SHMI. The URL hints at some climate issue, probably ozone.
2) What is the format?
   a) XML. The format is declared in the first line of the file.
   b) JSON. The format is identified by the browser in most cases nowadays.
   c) It's a CSV but instead of Comma Separated Values, the value are separatde by blank spaces.
3) What is the information contained?
   a) A database ID, a game name, the year of release, the platform that runs the game
   b) Various. But special database ID, band name and year of band fundation seems to be there. JSON is not a very structured format, it's just a collection of key-value pairs
   c) Hard to tell. The document lacks a description of the fields. The first column seem to be the year, the second the month, the third the day of the month according to the Swedish way of presenting dates. The rest of the values are hard to interpret.