AGENDA

- Introduction (Ulf)
- Structured data formats
  - Exercise
- Collecting and creating structured data
  - Exercise
- LUNCH
- Live demo – exploring data from the command line
  - Exercise
- Data management in practice
- Wrap-up and closing remarks (Ulf)

DATA

Structured Data Formats

AARHUS UNIVERSITY

19. AUGUST 2025

CHC

STRUCTURED DATA

- What is structured data?
  - A txt-file, csv-files, html, xml, json, SQL?
- Depends on the context
  - For humans or computers?
- Which format should we use for what?
- What formats can datasets created by others be expected to have?

```
<person>
  <name>Jane</name>
  <age>45</age>
  <salary>42000</salary>
</person>
```

```
Jane is a 45 years old and
has a salary of 42.000 per
month.
```

```
Jane, 45, 42000
```

```
{
  "name": "Jane",
  "age": 45,
  "salary": 42000
}
```

```
INSERT INTO persons
   VALUES("Jane", 45, 42000)
;
```

A BOOK IN TXT FORMAT

- For humans it seems structured
  - It has headings, paragraphs, chapters

For computers it is considered unstructured

  - It is just a sequence of characters
  - The txt format does not say anything about how to parse headings, paragraphs etc.

```
CHAPTER 1

Next to a great forest there lived a poor woodcutter with his
wife and his two children. The boy's name was Hansel and the
girl's name was Gretel. He had but little to eat, and once,
when a great famine came to the land, he could no longer
provide even their daily bread.

One evening as he was lying in bed worrying about his…
```

STRUCTURED DATA FOR COMPUTERS

- Requires a defined set of rules for how to parse the content of the file
- CSV/TSV
  - Row/column-based format
    - Row are separated by newline
    - Columns are separated by comma/tab
- HTML/XML
  - Tag based format where text and structure is surrounded by tags
  - `<TAG_NAME>some text</TAG_NAME>`
- JSON
  - Format based on JavaScript object syntax
  - Arrays, objects, strings, numbers, null
- Relational Databases
  - Tables, rows, columns
  - Relations using primary- and foreign keys

TABULAR DATA - CSV

- Often used for structuring research data
- We all have Excel or similar
- Can be imported from/exported to CSV from Excel
- CSV is easy to read (when viewed in Excel)
  - … and it just works in Excel 😐

Location	Temperature	Time
Aarhus	22	2023-05-12 15.00.23
Viborg	25	2023-05-12 14.10.54

```
Location,Temperature,Time
Aarhus,22,2023-05-12 15.00.23
Viborg,25,2023-05-12 14.10.54
```

CASE

- We want to find out what makes people happy looking at:
  - (name), age, salary, housing, civil status
- We want a structured and easy way to get an overview of our data
  - CSV seems to be the right choice

Name	Age	Salary	Housing	Civil Status	happiness
Jane	43	42.000	House	Relationship	0.7
John	34	31.000	Apartment	Married	0.5

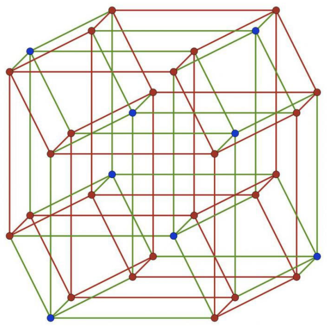
JOB DONE ☺

… wait! “I think that people's children have an impact on their happiness”

- We should add the children's **name** and **age**

ADDING MORE THAN 2 DIMENSIONS

- Possible solutions
  - Add multiple values in the same cell and create a custom rule for how to separate them
  - Add repeated columns for each new sub-domain of data
  - Repeat rows
  - Create a separate csv-file for each domain and link them using some kind of id



MULTIPLE VALUES IN SAME CELL

- Hard to filter/count/sort on the individual values in the cell
- We now have created a custom format
  - Requires instructions for how to parse the values in the cell
- Adding more dimensions will make it even harder to work with
  - … we need to define a new rule for each added dimension

Name	Age	Children
Jane	43	**Luke/Jill**
John	34	**Joe**

Name	Age	Children
Jane	43	**Luke:20/Jill:15**
John	34	**Joe:9**

REPEAT COLUMNS

- Hard to filter/count/sort on values in the cells as they span multiple columns
- We get a lot of empty cells
- Does not scale

Name	Age	Child1	Child2
Jane	43	**Luke**	**Jill**
John	34	**Joe**	

Name	Age	Child1-name	Child1-age	Child2-name	Child2-age
Jane	43	**Luke**	**20**	**Jill**	**15**
John	34	**Joe**	**9**		

REPEAT ROWS

- Hard to maintain as updates must be done in multiple rows
- We get a lot of redundant data

Name	Age	Child
Jane	43	**Luke**
Jane	43	**Jill**
John	34	**Joe**

NEW CSV FILE FOR EACH DOMAIN

- Hard to get and overview
- We have created a relational DB, but are missing the DBMS to query the data (make joins)

Persons

Id	Name	Age
1	Jane	43
2	John	34

Children

Id	Name	Age	ParentId
1	Luke	20	**1**
2	Jill	15	**1**
3	Joe	9	**2**

ALTERNATIVE SOLUTIONS

- XML
- JSON
- Relational DB (and SQL)

XML

```
<persons>
  <person name="Jane">
    <age>43</age>
    <salary>42000</salary>
    <housing>House</salary>
    <civilstatus>Relationship</civilstatus>
    <children>
      <child name="Luke">
        <age>20</age>
        <hobby name="Role Playing">
          <type>Acting</type>
        </hobby>
      </child>
       <child name="Jill">
        <age>15</age>
        <hobby name="Football">
          <type>Sport</type>
        </hobby>
      </child>
    </children>
  </person>
  <person name="John">
    <age>34</age>
    <salary>31000</salary>
    <housing>Apartment</salary>
    <civilstatus>Married</civilstatus>
    <children>
      <child name="Joe">
        <age>9</age>
        <hobby name="Football">
          <type>Sport</type>
        </hobby>
      </child>
    </children>
   </person>
</persons>
```

XML PROS AND CONS

- Pros
  - Text-based, ok to read
  - Good for detailed markup of text (words, phrases, paragraphs)
- Cons
  - Verbose format
  - requires extra information to determine types (xsd – Xml Schema Definition)
  - Creating new datasets can be a little hard by hand, requires an XML text-editor or a GUI
  - Can be difficult to avoid redundancy for repeated data
  - Are in many cases replaced by JSON

JSON

```
[
  {
    "name": "Jane",
    "age": 43,
    "salary": 43000,
    "housing": "House",
    "civilStatus": "Relationship",
    "children": [
      {
        "name": "Luke",
        "age": 20,
        "hobby": {
          "name": "Role Playing",
          "type": "Acting"
        }
      },
      { "name": "Jill", "age": 15, "hobby": { "name": "Football", "type": "Sport" } }
    ]
  },
  {
    "name": "John",
    "age": 34,
    "salary": 31000,
    "housing": "Apartment",
    "civilStatus": "Married",
    "children": [
      { "name": "Joe", "age": 9, "hobby": { "name": "Football", "type": "Sport" } }
    ]
  }
]
```

JSON STRUCTURE AND TYPES

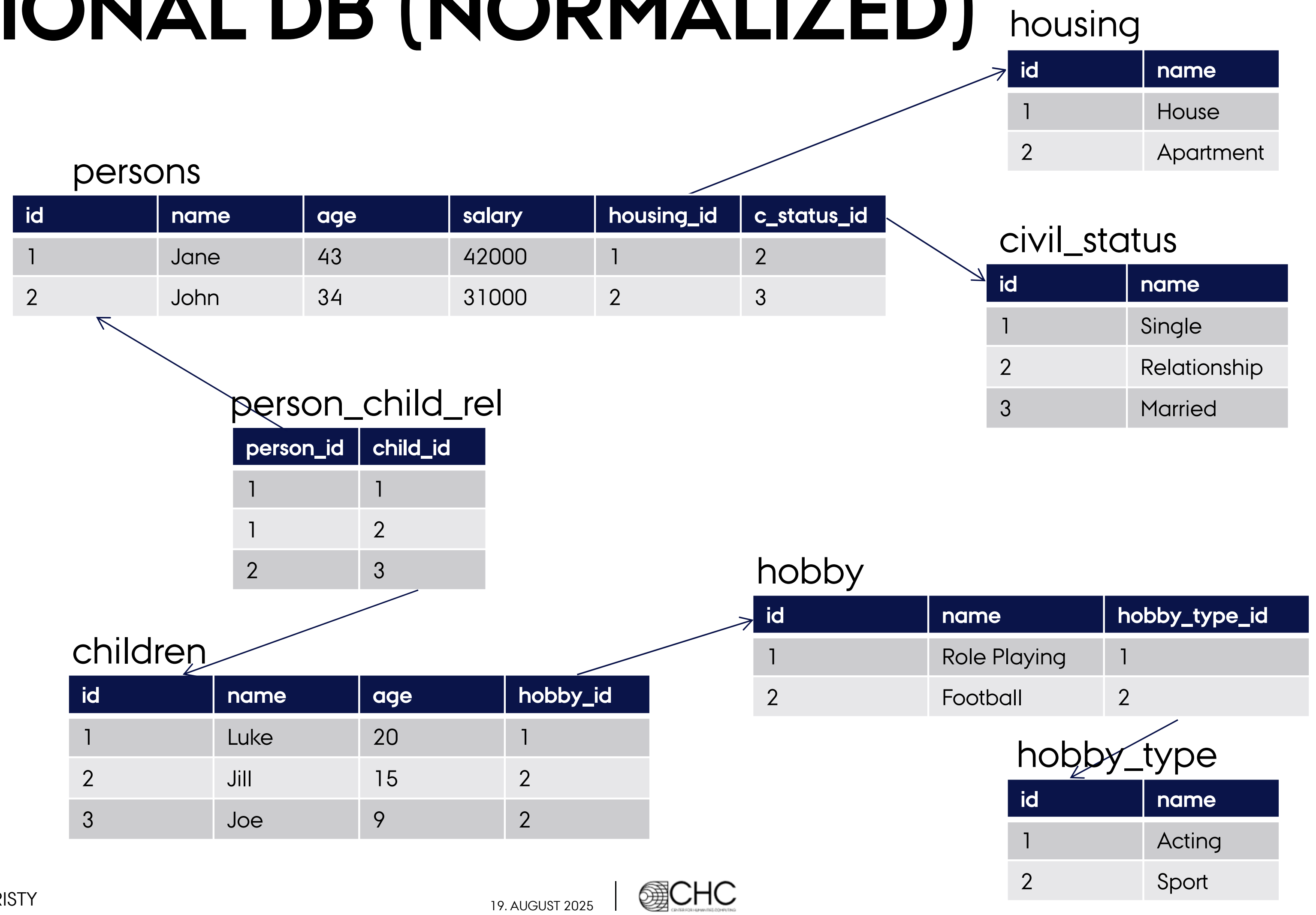
- Can start with an object or an array
- Objects and arrays can be nested indefinitely
- Property names must be surrounded by ""
- Arrays can contain any type. Each value must be separated by comma.

```
{
  "text": "text content",
  "number": 2.2,
  "boolean": true | false,
  "object": {},
  "array": []
  "null": null
}
```

JSON PROS AND CONS

- Pros
  - Text-based, easy to read
  - Simple and concise format
  - Easy to parse
  - Scales ok - for large datasets ndjson/jsonl can be used
- Cons
  - Creating new datasets can be a little hard by hand, requires a JSON text-editor or a GUI
  - Can be difficult to avoid redundancy for repeated data

RELATIONAL DB (NORMALIZED)



persons

id	name	age	salary	housing_id	c_status_id
1	Jane	43	42000	1	2
2	John	34	31000	2	3

housing

id	name
1	House
2	Apartment

civil_status

id	name
1	Single
2	Relationship
3	Married

person_child_rel

person_id	child_id
1	1
1	2
2	3

children

id	name	age	hobby_id
1	Luke	20	1
2	Jill	15	2
3	Joe	9	2

hobby

id	name	hobby_type_id
1	Role Playing	1
2	Football	2

hobby_type

id	name
1	Acting
2	Sport

RELATIONAL DB PROS AND CONS

- Pros
  - It can scale to very large datasets
  - Good for related data and for avoiding redundancy
    - Each piece of data is only registered once – makes it easier to maintain
    - Space efficient
  - Sqlite makes it possible to have a database in a single file
- Cons
  - Cannot be accessed without a DBMS, not directly human readable
  - Requires knowledge of SQL to edit and query the database
  - Requires relational DB modelling skills (normalization rules) if redundancy must be avoided

CHOOSING THE RIGHT FORMAT

- If your data has more than two dimension choose a format that supports that
  - Makes it easier to work with data
  - Can easily be used by others, because of standardization
- Do you need to add and edit parts of data and have repeated complex structures
  - Consider a relational database or JSON-objects with added ids and references
  - If you only need to read data, or can re-generate to whole datasets again as needed, relations are not required
- You can always transform from more to fever dimensions if required
  - E.g. from JSON -> CSV
  - https://json-to-csv.chc.au.dk/

EXERCISE

DATA

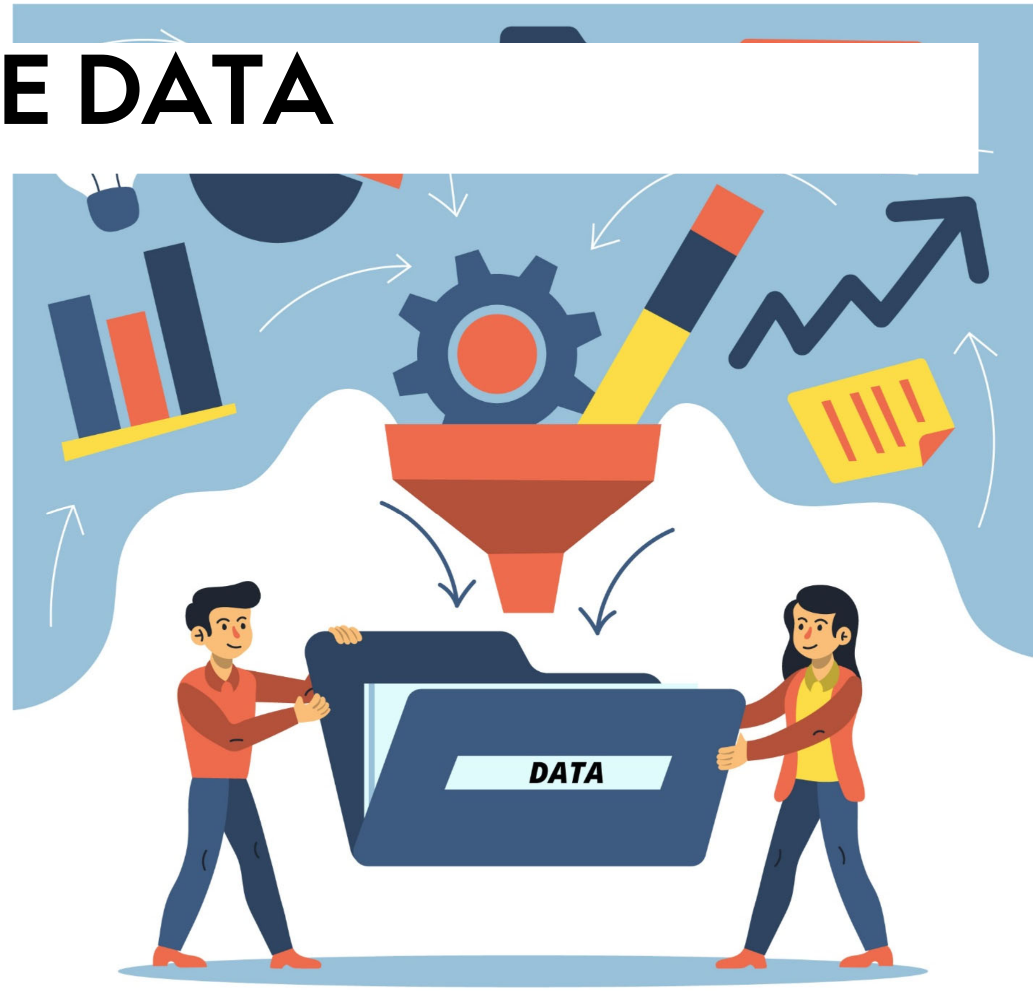
Collecting and Creating Structured Data

AARHUS UNIVERSITY

19. AUGUST 2025

HOW DO WE GET SOME DATA

- Create it ourselves based on findings, observations, interviews etc.
- Download an existing dataset
- Web-scraping
- AI assisted data extraction/generation

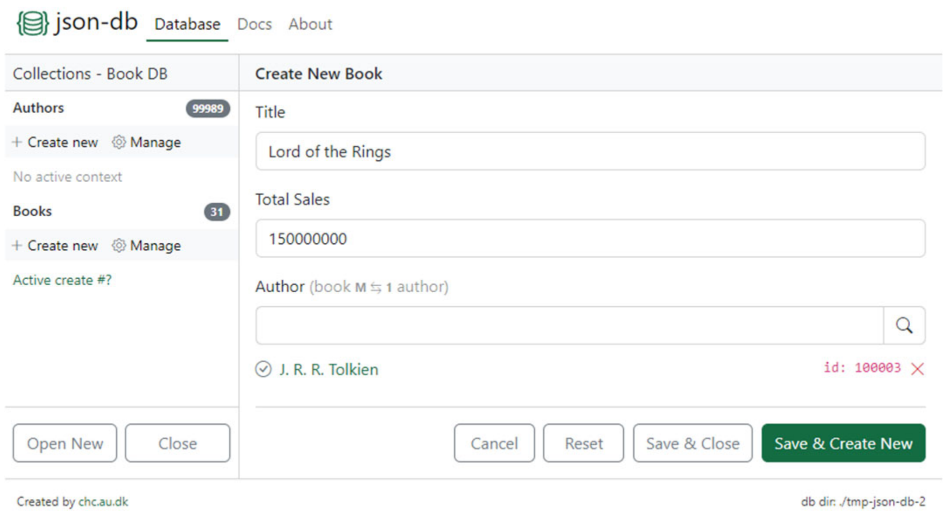


CREATE IT

- Tabular data
  - Use Excel or similar (we know how to do this)
- JSON
  - Use a JSON-editor, Notepad++, any code editor/IDE
  - Use a JSON database app, such as JSON-DB (next slide)
- Relational DB
  - CLI – requires good SQL skills
  - DB GUI – still requires SQL skills to filter and join tables
  - Custom made GUI – requires programming skills or costs money

JSON DB

- Editing JSON directly can for; large datasets, datasets with many properties and datasets with relational data, become challenging
- JSON-DB
  - A web-based offline database with a GUI
  - Reads and writes plain JSON files
  - Database structure is defined in a simple schema.json file
  - Edit existing data or create new data from scratch
  - Free to use
  - https://json-db.chc.au.dk/



Create New Book

Title: Lord of the Rings

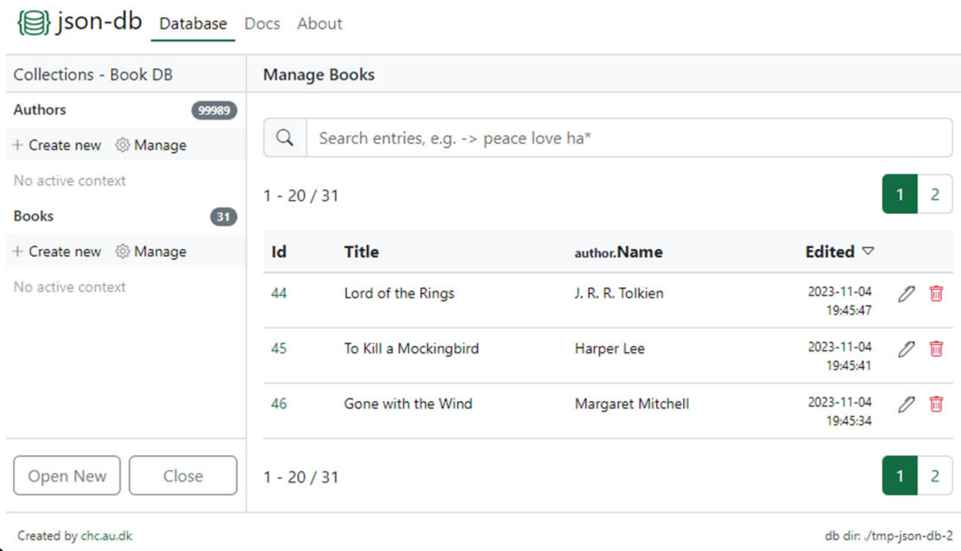
Total Sales: 150000000

Author (book M ⇆ 1 author)

J. R. R. Tolkien id: 100003

Created by chc.au.dk

db dir: ./tmp-json-db-2



Manage Books

Search entries, e.g. -> peace love ha*

1 - 20 / 31 · 1 2

Id	Title	author.Name	Edited
44	Lord of the Rings	J. R. R. Tolkien	2023-11-04 19:45:47
45	To Kill a Mockingbird	Harper Lee	2023-11-04 19:45:41
46	Gone with the Wind	Margaret Mitchell	2023-11-04 19:45:34

1 - 20 / 31 · 1 2

Created by chc.au.dk

db dir: ./tmp-json-db-2

SQLITE (SQL DATABASE)

- Sqlite is single file-based database
- The database CLI interface can easily be installed and runs on every OS
- Portable between systems
- Most used DB in the world – 1.000.000.000.000 active databases
- Many GUIs to inspect and create data
- Requires knowledge of SQL
- https://sqlite.org/
- https://sqlitebrowser.org/

```
CREATE TABLE persons (
  name VARCHAR(255) NOT NULL,
  age INTEGER NOT NULL,
  salary INTEGER NOT NULL
);
```

```
INSERT INTO TABLE persons VALUES
   ("Jane", 43, 42000)
   ("John", 34, 31000)
;
```

```
SELECT * FROM persons AS p
  WHERE p.salary > 35000
;
```

DOWNLOAD IT

- A lot of datasets already exists
  - E.g., KB, Twitter (X), SMK, Reddit, Infomedia etc.
  - See also https://kub.kb.dk/datalab/opendata
- Some can just be downloaded as files
- Others require access using an online API
  - Typically, a signup to get an access-token is required
  - Requires a little programming knowledge
  - Makes it possible to search and get live-data
  - Can cost money, e.g., Twitter (X)

```
import requests
import json


def fetch():
  url = "https://api.restful-api.dev/objects"

  response = requests.get(url)
  data = response.json()

  with open("result.json", mode="w", encoding="utf-8") as file:
    json.dump(data, file, ensure_ascii=False, indent=2)


if __name__ == "__main__":
    fetch()
```

WEB-SCRAPING

- The data is available, but only as HTML (website)
- Web-scraping is a programmatically technique for fetching and extracting data from a website
- Typical procedure:
  - Inspect the HTML using the browser-inspector to figure out the CSS-selector paths to use to get the data
  - Write a script to download the HTML-files of interest
  - Get and parse the HTML using, e.g., BeautifulSoup and requests (python)
  - Extract the relevant data using the CSS-selector paths from step 1
  - (Save the data to CSV/JSON)

```python
import requests
import bs4
import csv


def scrape():
  response = requests.get("https://dr.dk")
  soup = bs4.BeautifulSoup(response.text, "html.parser")

  headlines = [headline.text for headline in soup.select(".dre-teaser-title")]

  with open("headlines.csv", mode="w", newline="", encoding="utf-8") as file:
    writer = csv.writer(file)
    writer.writerow(["Headline"])
    for headline in headlines:
      writer.writerow([headline])


if __name__ == "__main__":
  scrape()
```

AI ASSISTED DATA EXTRACTION

- Prompt a GPT-model to make structured data from text, images and sound
- Beware: The model can make mistakes both in terms of “analysis” and structure of output
  - The large models are often trained on large sets of internet-data and can be biased
  - Furthermore, the model is typically tuned to answer or not answer in a certain way to not be harmful and make up for its biased data
  - Always be critical about the output and perform manual/trusted verification

```
From the below fairy tale do the following:
1. Find all characters
2. Determine if the character is Good, Neutral or Evil
3. Make a short description of the character
4. Create a JSON array with an object for each character in the format { name, alignment, description }
------
… FAIRY TALE HERE … e.g., https://www.cs.cmu.edu/~spok/grimmtmp/012.txt
```

EXERCISE