Data Formats, Interchange, and Processing

CCI BSC2: Data Visualisation and Sensing
Week 2
t.armitage@arts.ac.uk

Learning Objectives

- Understand some of the most commonly used file formats for data interchange
- Understand the need to parse and process that data prior to visualisation
- Consider ways of gathering and sourcing that data

Data Interchange

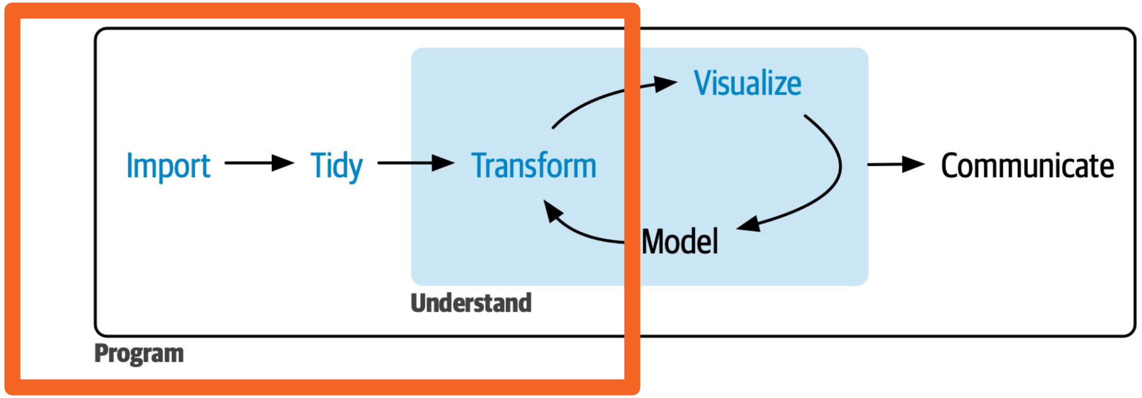
The formats we’re looking at today are all data *interchange* formats.

They’re not necessarily how the data is *stored*; instead, they are simple, standardised ways of transferring that data between platforms, programs, and use cases.not how data is stored, how it's shared/distributed

Or: they are *import/export* formats

Today’s formats are all plain-text, regular, easily processed, and have standard library support in almost all languages.

Isn't this a visualisation class?



Yes - but before we can visualise our data, we need to find it, source it, and possibly transform it for our purposes.

We need to understand the nature of our data in order to visualise it better.

This bit

Turning Data into Code

All these data files on disk are just a string of characters.

When we import it into our programming tool, we are **parsing** it - breaking it down into parts - and then transforming it into the **internal data structures** of our programming language.

In Python: lists, dicts (and some other things), made of strings, numbers, dates, etc..

In Javascript: arrays, objects, made of….

For any format:

- **Parse** it: turn it from a string of data into something in code you can work with.
- **Tidy** it: make sure it's in a good shape to work with. Make sure the things in it are the right type.
- **Explore** it: manipulate and discover meaning / relationships.
- **Transform** it: would it be easier to work with / visualise if there were new / other / different things in there?

Delimited Data

Delimited data

Data that’s separated by regular **delimiters** - things that define boundaries.

Common delimiters include: commas; tabs; other special characters (|)

Comma Separated Values - ***CSV*** - the most common, and what we will focus on.

CSV

CSV is an excellent representation of 2D, tabular data.

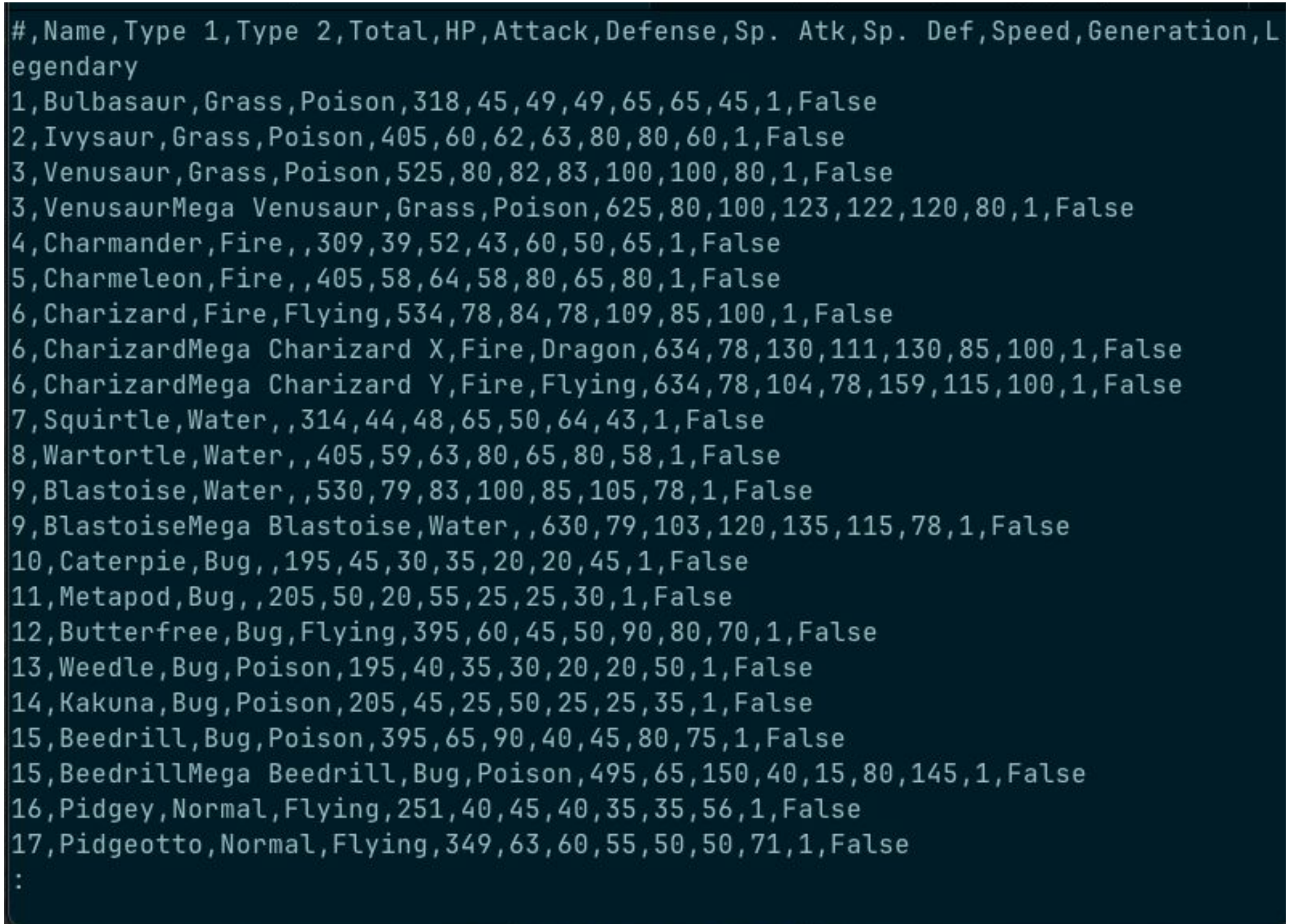
Rows are separated by newline characters (\n)

Fields/cells are separated by commas.

Fields that should include a comma are “wrapped in quote marks”.

A common “export” format from spreadsheet tools (Excel, Numbers, Sheets) - note that ONLY data will be exported, not formulae.

CSV



```
#,Name,Type 1,Type 2,Total,HP,Attack,Defense,Sp. Atk,Sp. Def,Speed,Generation,L
egendary
1,Bulbasaur,Grass,Poison,318,45,49,49,65,65,45,1,False
2,Ivysaur,Grass,Poison,405,60,62,63,80,80,60,1,False
3,Venusaur,Grass,Poison,525,80,82,83,100,100,80,1,False
3,VenusaurMega Venusaur,Grass,Poison,625,80,100,123,122,120,80,1,False
4,Charmander,Fire,,309,39,52,43,60,50,65,1,False
5,Charmeleon,Fire,,405,58,64,58,80,65,80,1,False
6,Charizard,Fire,Flying,534,78,84,78,109,85,100,1,False
6,CharizardMega Charizard X,Fire,Dragon,634,78,130,111,130,85,100,1,False
6,CharizardMega Charizard Y,Fire,Flying,634,78,104,78,159,115,100,1,False
7,Squirtle,Water,,314,44,48,65,50,64,43,1,False
8,Wartortle,Water,,405,59,63,80,65,80,58,1,False
9,Blastoise,Water,,530,79,83,100,85,105,78,1,False
9,BlastoiseMega Blastoise,Water,,630,79,103,120,135,115,78,1,False
10,Caterpie,Bug,,195,45,30,35,20,20,45,1,False
11,Metapod,Bug,,205,50,20,55,25,25,30,1,False
12,Butterfree,Bug,Flying,395,60,45,50,90,80,70,1,False
13,Weedle,Bug,Poison,195,40,35,30,20,20,50,1,False
14,Kakuna,Bug,Poison,205,45,25,50,25,25,35,1,False
15,Beedrill,Bug,Poison,395,65,90,40,45,80,75,1,False
15,BeedrillMega Beedrill,Bug,Poison,495,65,150,40,15,80,145,1,False
16,Pidgey,Normal,Flying,251,40,45,40,35,35,56,1,False
17,Pidgeotto,Normal,Flying,349,63,60,55,50,50,71,1,False
:
```

CSV: what's in it, what isn't.

The first line of a CSV file is usually (not always) the column headers, that describe what else is in it.

There are lots of things CSV files **can't** contain:

- **Object types.** What's a string? What's a number? How would you know?
- **Documentation/comments.** What you're getting is just data. (Documentation might be in a separate file).
- **Nested relations.** CSV is limited to two dimensions, and that's it.

Tools for exploring CSV

Lots of tools can process CSV files:

- **Spreadsheet tools** can also import CSV. However: they are **not** always very performant with very large data files, and not always the best choice for quick manipulation.
- Simple UNIX **command-line tools** will often give you a very quick heads-up (sorry, Windows folks) - head, tail, less, wc
- **Standard libraries** for most programming languages will let you open something up quickly in a REPL/console/notebook
- **Dedicated data-analysis packages** or languages (R, SAS, pandas) will have excellent CSV handling tools

Examples

- View file in terminal
- View file in text editor
- Quick Python `csv` examples.
- Pandas examples

Other delimiters are available!

Tabs, commonly.

Special characters.

Other text formats are available - fixed-width (every cell has the same spacing); custom, strange binary formats.

Some have libraries. All should have documentation. Read the docs for a dataset if it comes with them!

JSON

JSON

Java**S**cript **O**bject **N**otation

Very common these days.

More sophisticated than CSV:

- Multiple data types: strings, numbers, lists, objects
- Hierarchical - can contain multiple dimensions (an object can contain objects)

Most languages have a core library for it, not just javascript

JSON

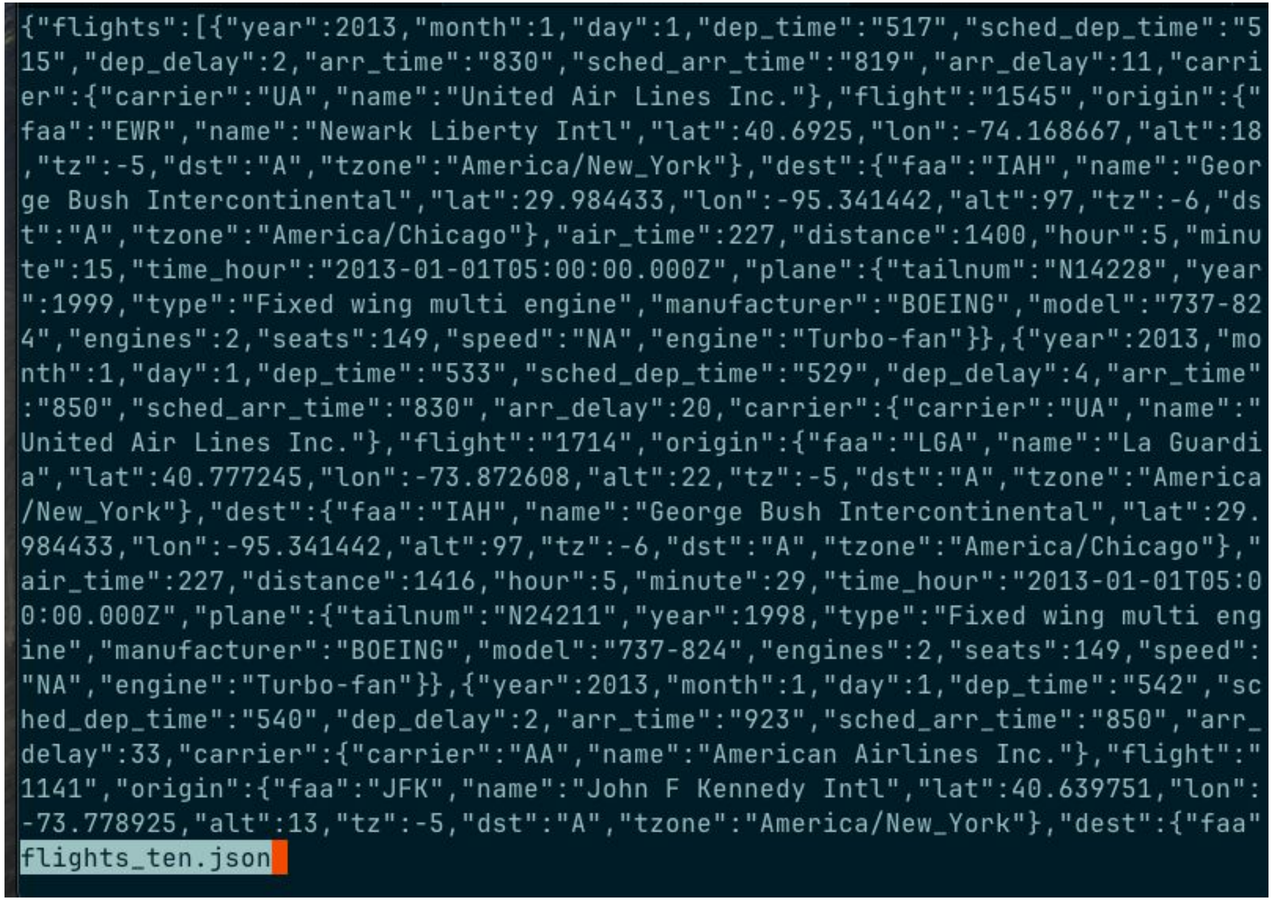
Well suited to nested / deep data

Well suited to heterogeneous data (objects can have different / missing headings)

Particularly well suited to the web: excellent support inside Javascript and the browser (unsurprisingly)

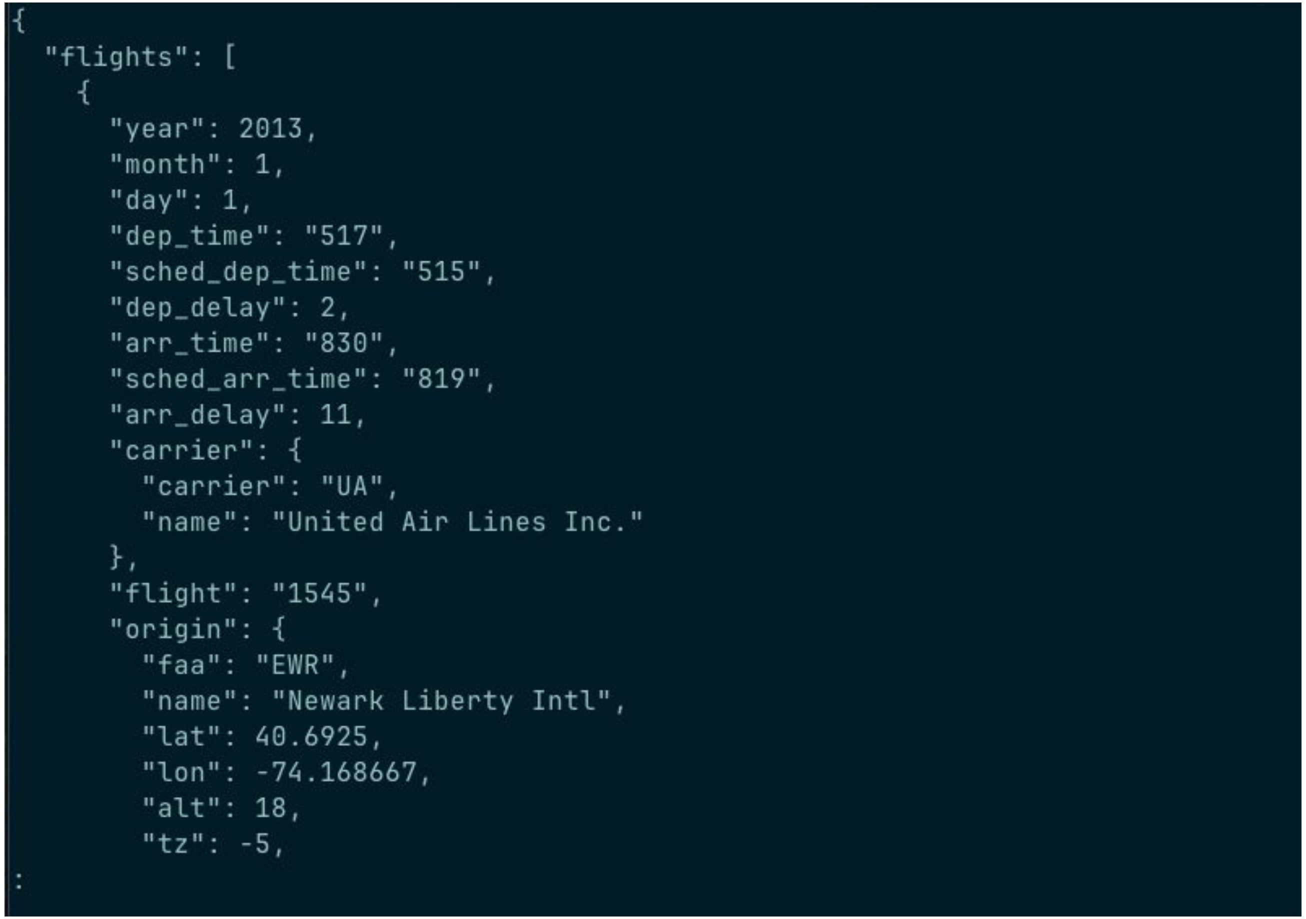
As a result of this, **very** common as an API interchange format (we'll come back to this)

JSON



```
{"flights":[{"year":2013,"month":1,"day":1,"dep_time":"517","sched_dep_time":"5
15","dep_delay":2,"arr_time":"830","sched_arr_time":"819","arr_delay":11,"carri
er":{"carrier":"UA","name":"United Air Lines Inc."},"flight":"1545","origin":{"
faa":"EWR","name":"Newark Liberty Intl","lat":40.6925,"lon":-74.168667,"alt":18
,"tz":-5,"dst":"A","tzone":"America/New_York"},"dest":{"faa":"IAH","name":"Geor
ge Bush Intercontinental","lat":29.984433,"lon":-95.341442,"alt":97,"tz":-6,"ds
t":"A","tzone":"America/Chicago"},"air_time":227,"distance":1400,"hour":5,"minu
te":15,"time_hour":"2013-01-01T05:00:00.000Z","plane":{"tailnum":"N14228","year
":1999,"type":"Fixed wing multi engine","manufacturer":"BOEING","model":"737-82
4","engines":2,"seats":149,"speed":"NA","engine":"Turbo-fan"}},{"year":2013,"mo
nth":1,"day":1,"dep_time":"533","sched_dep_time":"529","dep_delay":4,"arr_time"
:"850","sched_arr_time":"830","arr_delay":20,"carrier":{"carrier":"UA","name":"
United Air Lines Inc."},"flight":"1714","origin":{"faa":"LGA","name":"La Guardi
a","lat":40.777245,"lon":-73.872608,"alt":22,"tz":-5,"dst":"A","tzone":"America
/New_York"},"dest":{"faa":"IAH","name":"George Bush Intercontinental","lat":29.
984433,"lon":-95.341442,"alt":97,"tz":-6,"dst":"A","tzone":"America/Chicago"},"
air_time":227,"distance":1416,"hour":5,"minute":29,"time_hour":"2013-01-01T05:0
0:00.000Z","plane":{"tailnum":"N24211","year":1998,"type":"Fixed wing multi eng
ine","manufacturer":"BOEING","model":"737-824","engines":2,"seats":149,"speed":
"NA","engine":"Turbo-fan"}},{"year":2013,"month":1,"day":1,"dep_time":"542","sc
hed_dep_time":"540","dep_delay":2,"arr_time":"923","sched_arr_time":"850","arr_
delay":33,"carrier":{"carrier":"AA","name":"American Airlines Inc."},"flight":"
1141","origin":{"faa":"JFK","name":"John F Kennedy Intl","lat":40.639751,"lon":
-73.778925,"alt":13,"tz":-5,"dst":"A","tzone":"America/New_York"},"dest":{"faa"
flights_ten.json
```

JSON



```
{
  "flights": [
    {
      "year": 2013,
      "month": 1,
      "day": 1,
      "dep_time": "517",
      "sched_dep_time": "515",
      "dep_delay": 2,
      "arr_time": "830",
      "sched_arr_time": "819",
      "arr_delay": 11,
      "carrier": {
        "carrier": "UA",
        "name": "United Air Lines Inc."
      },
      "flight": "1545",
      "origin": {
        "faa": "EWR",
        "name": "Newark Liberty Intl",
        "lat": 40.6925,
        "lon": -74.168667,
        "alt": 18,
        "tz": -5,
:
```

JSON

Both these files are the same to the computer.

One is a bit easier for us.

In general: rely on your **tools** to do the formatting, don’t necessarily worry about what the file looks like.

JSON Examples

- View file in terminal
- View file in text editor
- Parse in Python

Other structured data formats are available!

XML, notably - prior to JSON, probably the most common structured format.

Very similar in terms of capabilities; looks more complex written down, but you’re using a parser/library so that doesn’t matter.

File formats vs data formats

CSV and JSON are **file formats** but the uses they are put to are many.

However, there **are** standardised **data formats** built out of this.

Eg: GeoJSON is a **standard** for representing geographical information as JSON. It’s written in JSON, but there are specific rules for its **structure**.

Formats are also documented - and there may be dedicated libraries for them.

Most data formats are unique

Often, your CSV files will come with **documentation**, explaining the headings, what the enumerations are. Those docs are for you! You can't expect to understand something just by looking at numbers. The documentation might also explain domain-specific details (what categories mean, for instance).

Data acquisition

Data acquisition

Where can we get data from? How can we work with it?

Public downloads

Eg:

- https://github.com/awesomedata/awesome-public-datasets
- https://www.data.gov.uk/
- https://data.gov/

Be aware of how well text compresses!

Example

- Camden Air Quality Data
- https://opendata.camden.gov.uk/Environment/London-Air-Quality-Network-Camden/gc7n-mvxa/about_data

Public APIs

API == Application Programming Interface - a specified interface to write code against.

“Public” API = one anyone can get access to, via the internet.

Often used for data retrieval.

HTTP APIs

The most common approach to a public API is to use HTTP:

- Make a request (possibly with some parameters) over HTTP to an endpoint (URL)
- Get data back.

The requests you make determine what you get back

Just because it's HTTP doesn't mean it happens in a browser! HTTP has become the de-facto transit layer for data APIs.

We can also use code/cli to do this, eg `requests` library

HTTP API: demo

A short demo of using *requests* to access Flights data over an API.

The code for this notebook, and the API itself, are on git.arts.ac.uk

API Limits

Public APIs will have limits and constraints on their usage.

- **Authentication** - you will often need a KEY, a secret code you pass to the endpoint, to identify yourself, and possibly authorize what you can do
- **Rate limiting** - you could have a limit on how many requests you can make a minute/second before you are rejected/have to back off.
- **Terms of service restrictions** - you may be restricted in how you can use the data you acquire.

Read the documentation for your API.

Scraping

“Scraping” = “reading the HTML of webpages with a machine to extract data from them”.

If a webpage is structured markup - which it is - we can use a parser to extract data from it, just like we can parse XML.

Scraping: parsers

- Feed the contents of a page to a parser (eg: BeautifulSoup)
- If you can work out how to describe the location of what you'd like, you can get that data out
- for instance: "*find the table with id **countries**, and get every row in it*"; then, get every cell in the row
- Transform string values to integers and off you go.
- Identify elements via **selectors** just like in CSS: tag, id, attributes, dataset
- HTML is *very* permissive. Good parsers (like BS) cope with this.

Limits of Scraping

Very useful for working with data that's not easily available. Often used as a bodge/hack/workaround. **But:**

- Gets harder the worse the page HTML is (and page HTML in 2025 is very bad)
- Needs rewriting whenever the page changes
- Robots are blocked a *lot* in 2025
- You need to be a good citizen: proper user-agent strings, reasonable rate limits, back off when throttled, etc, etc.
- Very fragile.

If you REALLY think you need to scrape things for your project

Chat to me in the post-class time one week: there might be a tutorial we can organise around scraping, or alternative approaches.

?

Data Formats, Interchange, and Processing

CCI BSC2: Data Visualisation and Sensing
Week 2
t.armitage@arts.ac.uk