Applied Data Analysis (CS401)

EPFL

Lecture 2
Handling data
17 Sep 2025

Maria Brbić

Announcements

- **Register** your teams (5 people) by **Fri Sep 26th**
  - https://go.epfl.ch/ada2025-team-registration
  - Each team member must individually complete the form!
- **Project milestone P1** to be released this Fri, due **Wed Oct 1st**
- **First quiz** ("Q1"*) will be released today after the lecture
  - Exercise for recap lecture materials
- **Friday's lab session:**
  - Intro to Pandas (very important for Homework H1 and rest of course)
  - Exercise 1 already on Github

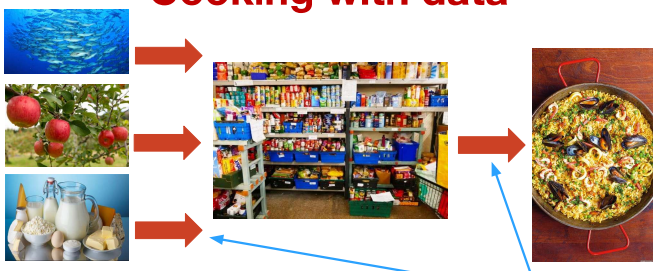
Feedback

Give us feedback on this lecture here:
https://go.epfl.ch/ada2025-lec2-feedback

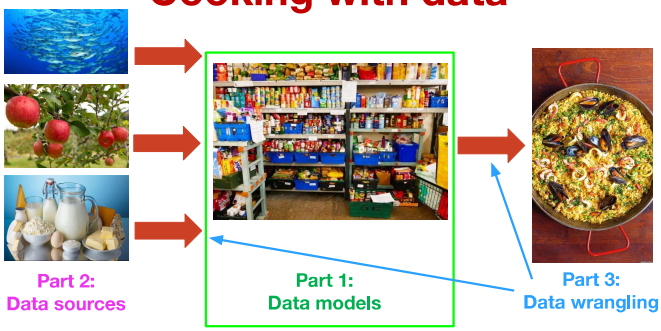
Feedback form available for each lecture and lab session

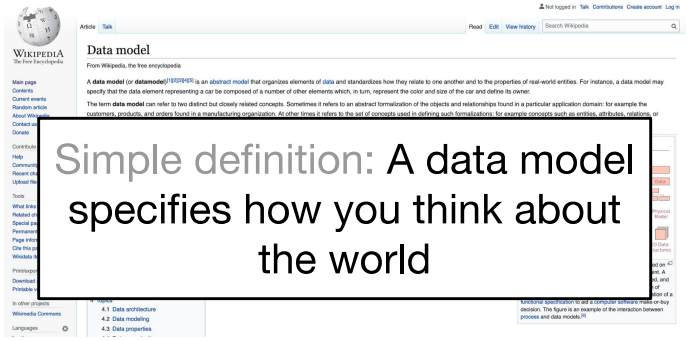
- What did you (not) like about this lecture?
- What was (not) well explained?
- On what would you like more (fewer) details?
- Is it nicer to follow the lecture online or offline?
- …

Cooking with data

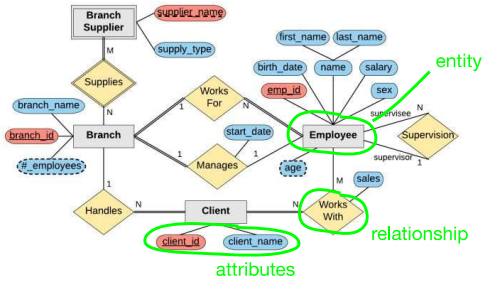


Cooking with data

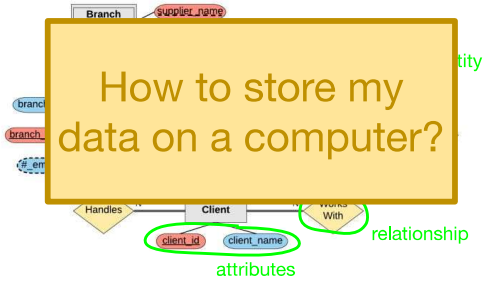




Q: “How do you think about the world?”
A: “See my entity–relationship diagram!”

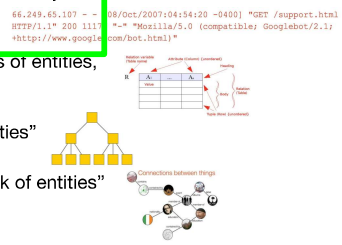


Q: “How do you think about the world?”
A: “See my entity–relationship diagram!”



Q1: “How should I store my data on a computer?”
Q2: “How do I think about the world?”

- “The world is simple: one type of entity, all with the same attributes”
  → **Flat model**
- “The world contains many types of entities, connected by relationships”
  → **Relational model**
- “The world is a hierarchy of entities”
  → **Document model**
- “The world is a complex network of entities”
  → **Network model**



Flat model

“The world is simple: one type of entity, all with the same attributes”

- Example: log files; e.g., Apache web server (`httpd`)
  - Entities = requests from clients to server

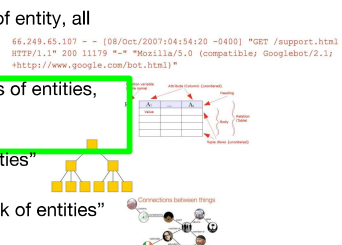
```
66.249.65.107 - - [08/Oct/2007:04:54:20 -0400] "GET /support.html
HTTP/1.1" 200 11179 "-" "Mozilla/5.0 (compatible; Googlebot/2.1;
+http://www.google.com/bot.html)"

111.111.111.111 - - [08/Oct/2007:11:17:55 -0400] "GET / HTTP/1.1" 200
10801
"http://www.google.com/search?q=in+love+with+ada+lovelace+what+to+do&ie=ut
f-8&oe=utf-8&aq=t&rls=org.mozilla:en-US:official&client=firefox-a"
"Mozilla/5.0 (Windows; U; Windows NT 5.2; en-US; rv:1.8.1.7)
```

- Another common format: CSV (“comma-separated vector”)

Q1: “How should I store my data on a computer?”
Q2: “How do I think about the world?”

- “The world is simple: one type of entity, all with the same attributes”
  → **Flat model**
- “The world contains many types of entities, connected by relationships”
  → **Relational model**
- “The world is a hierarchy of entities”
  → **Document model**
- “The world is a complex network of entities”
  → **Network model**



Relational model

“The world contains many types of entities, connected by relationships.”

- The relational model is ubiquitous:
  - MySQL, PostgreSQL, Oracle, DB2, SQLite, …
  - You use it many times every day
- Data represented as tables describing
  - entities
  - relationships between entities

id	name
1	Bush
2	Trump
3	Obama

president	successor
1	3
3	2

Processing data in the relational model: SQL

SQL (Structured Query Language)

- *Declarative* language for core data manipulations
- You think about what you want, not how to compute it

Imperative (e.g., Python)

```
//dogs = [{name: 'Fido', owner_id: 1}, {...}, ... ]
//owners = [{id: 1, name: 'Bob'}, {...}, ...]

var dogsWithOwners = []
var dog, owner

for(var di=0; di < dogs.length; di++) {
  dog = dogs[di]

  for(var oi=0; oi < owners.length; oi++) {
    owner = owners[oi]
    if (owner && dog.owner_id == owner.id) {
      dogsWithOwners.push({
        dog: dog,
        owner: owner
      })
    }
  }
}
```

Declarative (e.g., SQL)

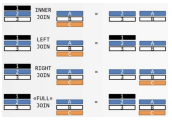
```
SELECT * from dogs
INNER JOIN owners
WHERE dogs.owner_id = owners.id
```

From: http://latentflip.com/imperative-vs-declarative/

SQL

```
SELECT * from dogs
INNER JOIN owners
WHERE dogs.owner_id = owners.id
```

- You should know basics of SQL
- Need a refresher? → Watch/do online tutorials!
- Key operations:
  - Select (!), update, delete
  - Unique keys
  - Joins (inner, left outer, right outer, full)
  - Sorting
  - Aggregation (group by, count, min, max, avg, etc.)



POLLING TIME

- **"Have you worked with SQL joins?"**
- Scan QR code or go to https://app.sli.do/event/jiFfkrP812UjtpwMbFJNxD

SQL implementations

MySQL

SQLite

etc.

```
#!/usr/bin/python

import MySQLdb

# Open database connection
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

sql = "SELECT * FROM EMPLOYEE \
       WHERE INCOME > '%d'" % (1000)
try:
   # Execute the SQL command
   cursor.execute(sql)
   # Fetch all the rows in a list of lists.
   results = cursor.fetchall()
   for row in results:
      fname = row[0]
      lname = row[1]
      age = row[2]
      sex = row[3]
      income = row[4]
      # Now print fetched result
      print "fname=%s,lname=%s,age=%d,sex=%s,income=%d" % \
            (fname, lname, age, sex, income )
except:
   print "Error: unable to fecth data"

# disconnect from server
db.close()
```


SQL and "SQL"

- The declarative-programming principles of SQL are widespread, even where it's less obvious

"SQL": Pandas (Python library)

- Similar to SQL (declarative), with additional elements of functional programming (map(), filter(), etc.)
- SQL "table" ←→ Pandas "DataFrame"
- Need intro? Come to Friday's lab session!

Pandas vs. SQL

+ Pandas is lightweight and fast
+ Natively Python, i.e., full SQL expressiveness plus the expressiveness of Python, especially for function evaluation
+ Integration with plotting functions like Matplotlib

- In Pandas, tables must fit into memory
- No post-load indexing functionality: indices are built when a table is created
- No transactions, journaling, etc. (matters for parallel applications)
- Large, complex joins are slower

"SQL": Unix command line

Goal: Find top 5 URLs visited most frequently by users between 18 and 25 years old

users.txt

user_id	age
User145	33
User24	15
User5	76
...	

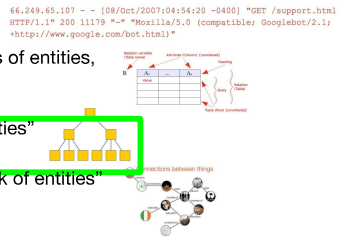
url_visits.txt

user_id	url
User2	ada.epfl.ch
User244	facebook.com
...	

```
cat users.txt \
| awk '$2 >= 18 && $2 <= 25' \
| join -1 1 -2 1 url_visits.txt - \
| cut -f 4 \
| sort \
| uniq -c \
| sort -k 1,1 -n -r \
| head -n 5
```

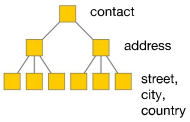

Q1: "How should I store my data on a computer?" Q2: "How do I think about the world?"

- "The world is simple: one type of entity, all with the same attributes"
  → **Flat model**
- "The world contains many types of entities, connected by relationships"
  → **Relational model**
- "The world is a hierarchy of entities"
  → **Document model**
- "The world is a complex network of entities"
  → **Network model**



Document model

"The world is a hierarchy of entities"



- XML format:

```
<contact>
  <id>656</id>
  <firstname>Chuck</firstname>
  <lastname>Smith</lastname>
  <phone>(123) 555-0178</phone>
  <phone>(890) 555-0133</phone>
  <address>
   <street>Rue de l'Ale 8</street>
   <city>Lausanne</city>
   <zip>1007</zip>
   <country>CH</country>
  </address>
</contact>
```

- JSON format:

```
contact: {
  id: 656,
  firstname: "Chuck",
  lastname: "Smith",
  phones: ["(123) 555-0178",
           "(890) 555-0133"],
  address: {
   street: "Rue de l'Ale 8",
   city: "Lausanne",
   zip: 1007,
   country: "CH"
  }
}
```

"Think for a minute" 🤔

- Document model

```
<contact>
  <id>656</id>
  <firstname>Chuck</firstname>
  <lastname>Smith</lastname>
  <phone>(123) 555-0178</phone>
  <phone>(890) 555-0133</phone>
  <address>
   <street>Rue de l'Ale 8</street>
   <city>Lausanne</city>
   <zip>1007</zip>
   <country>CH</country>
  </address>
</contact>
```

Think for a minute:

If we want to use a relational DB (e.g., MySQL) instead of XML, how can we store several phone numbers for the same person?

(Feel free to discuss with your neighbor.)

Solution to "Think for a minute"

- Document model

```
<contact>
  <id>656</id>
  <firstname>Chuck</firstname>
  <lastname>Smith</lastname>
  <phone>(123) 555-0178</phone>
  <phone>(890) 555-0133</phone>
  <address>
   <street>Rue de l'Ale 8</street>
   <city>Lausanne</city>
   <zip>1007</zip>
   <country>CH</country>
  </address>
</contact>
```

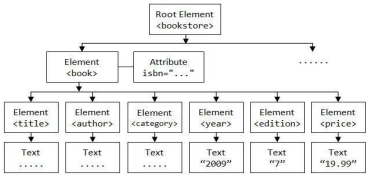
- Same in relational model

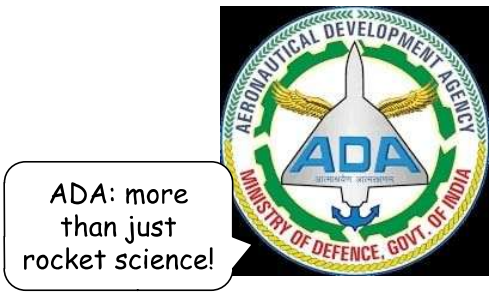
id	first name	...
656	Chuck	...
...	...	...

id	phone
656	(123) 555-0178
656	(890) 555-0133
...	...

Processing XML and JSON

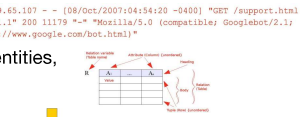
- Document structure = tree
- Processing via tree traversal (depth- or breadth-first search)
- Or use proper query language, such as XQuery or jq





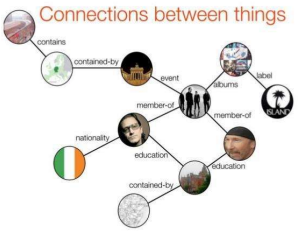
Q1: "How should I store my data on a computer?"
Q2: "How do I think about the world?"

- "The world is simple: one type of entity, all with the same attributes"
  → **Flat model**
- "The world contains many types of entities, connected by relationships"
  → **Relational model**
- "The world is a hierarchy of entities"
  → **Document model**
- "The world is a complex network of entities"
  → **Network model**



Network model

"The world is a complex network of entities"

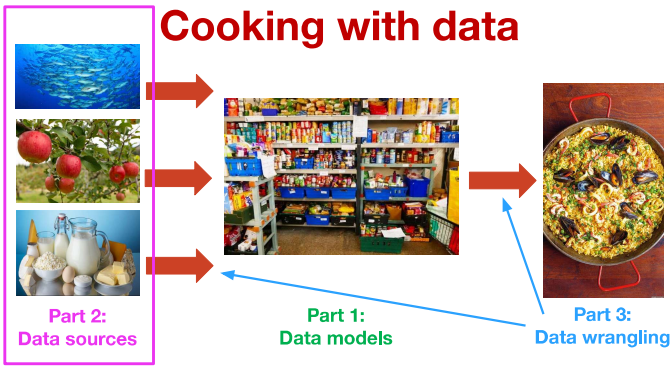


"How should I store my data on a computer?" —A word (or two) on binary formats

- "Parsing" = converting strings (as stored in text files) to data types used by computer programs (e.g., int, float, boolean, array, list)
- Possibly expensive, but can be avoided by using binary formats: store data to disk "as is", without first converting to strings
- Modern binary formats support nested structures, various levels of schema enforcement, compression, etc.
- Python pickle, Java Serializable, Protocol Buffers (Google), Avro (supports schema evolution), Parquet (column-oriented), etc.

→ Consider converting to a binary format at the beginning of your processing pipeline (especially when using "big data")

Cooking with data



Data sources at Web companies

Examples from Facebook / Meta

- Application databases
- Web server logs
- Client-side event logs
- API server logs
- Ad server logs
- Search server logs
- Advertisement landing page content
- Wikipedia
- Images and video

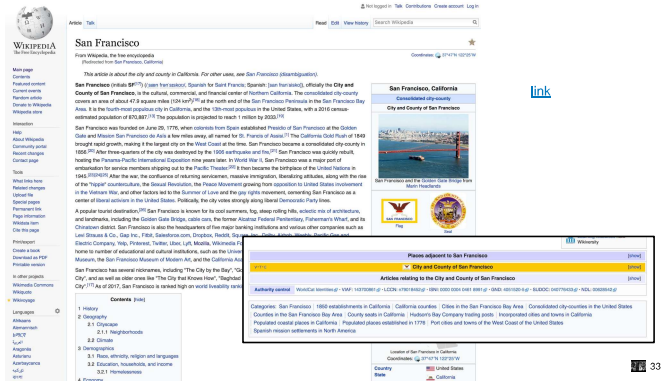
Structured data (with clear schema)

Semi-structured data ("self-describing" structure; CSV etc.)

Unstructured data

Another example: Wikipedia

- 300+ languages
- 42 million entities
- Mind-boggling richness of data



link

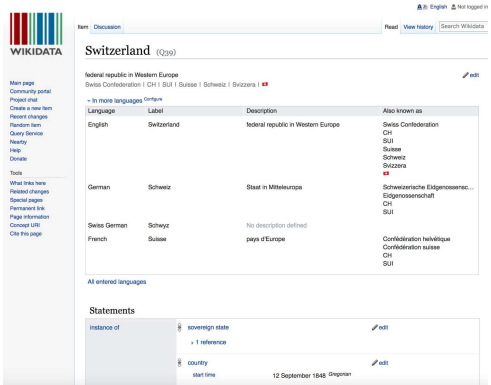
Wikipedia

How to work with Wikipedia?

- REST (cf. later) API
- XML dumps with wiki markup, SQL database dumps
- Issues: Unicode, size, recency, etc.
- To make your life easier:
  (1) Find projects on GitHub to help you
  (2) Use more structured versions (p.t.o.)

Wikidata

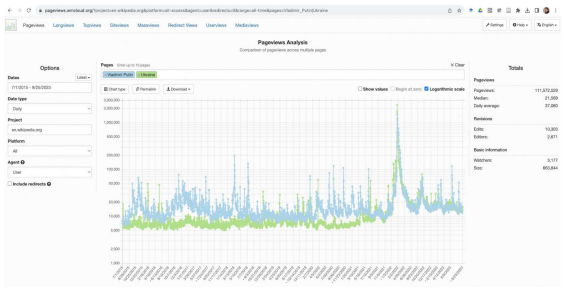
- "Database version" of Wikipedia
- {fr:Suisse, de:Schweiz, it:Svizzera, en:Switzerland, …} → Q39



Wikidata

- "Database version" of Wikipedia
- {fr:Suisse, de:Schweiz, it:Svizzera, en:Switzerland, …} → Q39
- Both API access and full database dumps
- Available as
  - JSON (document model)
  - RDF (network model)

Wikipedia pageview logs



Link:
https://pageviews.wmcloud.org/?project=en.wikipedia.org&platform=all-access&agent=user&redirects=0&range=all-time&pages=Vladimir_Putin|Ukraine

Crawling and processing webpages: HTML

Plenty of bulk-downloadable HTML data:

- Common Crawl dataset, about 1.82 billion web pages -- huge!
- (… but less than 0.1% of Google's Web crawl, as of 2015)
- 145 TB, hosted on Amazon S3, also available for download

… but if you need a specific website: use a **crawler/"spider"**: Apache Nutch, Storm, Heritrix 3, Scrapy, etc. (or simply wget…)

Useful HTML tools

Requests http://docs.python-requests.org/en/master/
An elegant and simple HTTP library for Python

Scrapy https://scrapy.org/
An open-source framework to build Web crawlers

Beautiful Soup http://www.crummy.com/software/BeautifulSoup/
A Python API for handling real HTML

Plain ol' /regular/expres*ion/s…

Schema.org: microformats for Web pages

- Nuggets of structured information embedded in (semantically) unstructured HTML

Text as rendered by browser:

Avatar
Director: James Cameron (born August 16, 1954)
…

HTML under the hood:

```
<div itemscope itemtype="http://schema.org/Movie">
  <h1 itemprop="name">Avatar</h1>
  <div itemprop="director" itemscope itemtype="http://schema.org/Person">
  Director: <span itemprop="name">James Cameron</span>
(born <time itemprop="birthDate" datetime="1954-08-16">August 16, 1954</time>)
  </div>
  <span itemprop="genre">Science fiction</span>
  <a href="../movies/avatar-theatrical-trailer.html" itemprop="trailer">Trailer</a>
</div>
```


Web services

- Most large web sites today actively discourage "screen-scraping" to get their content
- Instead: Web service APIs, for interoperable machine-to-machine interaction over a network
- The preferred way to get data from online sources
- Most common framework: REST
  - You request a URL from the server via HTTP
  - The server responds with a text file (e.g., JSON, XML, plain text)

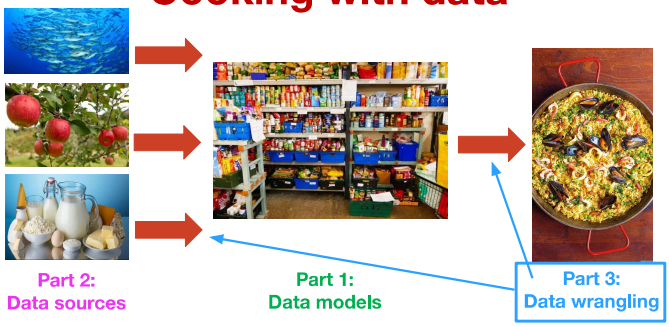
REST example

```
{
  "user": {
    "name": "Jane",
    "gender": "female",
    "location": {
      "href":
"http://www.example.org/us
/ny/new_york",
      "text": "New York"
    }
  }
}
```

← This resource is a description of a user named Jane

- Requested by sending GET request for the resource's URL, e.g., via curl:
  `curl http://www.example.org/users/jane/`
- If users need to modify the resource, they GET it, modify it, and PUT it back
- The href to the location resource allows savvy clients to get more information with another simple GET request
- Implication: Clients cannot be too "thin"; need to understand resource formats!

Cooking with data



Working with raw data sucks

Data comes in all shapes and sizes
– CSV files, PDFs, SQL dumps, .jpg, …

Different files have different formatting
– Empty string or space instead of NULL, extra header rows, character encoding, …

"Dirty" data
– Unwanted anomalies, duplicates

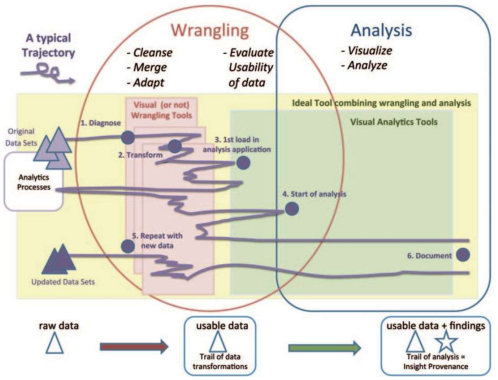
Raw data without thinking: A recipe for disaster!

What is data wrangling?

- **Goal:** extract and standardize the raw data
  - Combine multiple data sources
  - Clean data anomalies
- **Strategy:** Combine automation with interactive visualizations to aid in cleaning

Wrangling takes **between 50% and 80% of your time…**

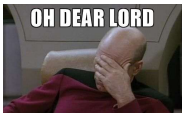
[Source]



Types of data problems

- Missing data
- Incorrect data
- Inconsistent representations of the same data
- About 75% of data problems require human intervention (e.g., experts, crowdsourcing, etc.)
- Tradeoff between cleaning data vs. over-sanitizing data


"Dirty data" horror stories



"Dear Idiot" letter

17,000 men are pregnant

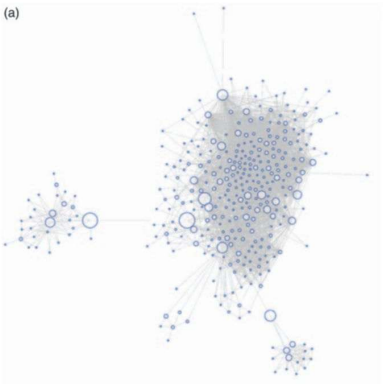
As the crow flies

[Source]

Diagnosing data problems

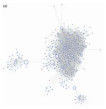
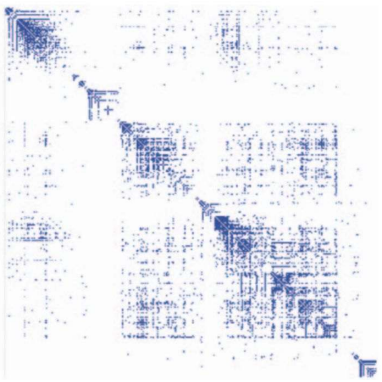
- Visualizations and basic stats can convey issues in "raw" data
- Different representations highlight different types of issues:
  - Outliers often stand out in the right kind of plot
  - Missing data will cause gaps or zero values in the right kind of plot
- Becomes increasingly difficult as data gets larger
  - Sampling to the rescue!

Facebook graph



Matrix view (1)

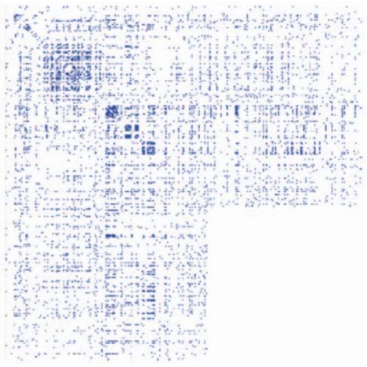
Automatic permutation of rows and columns to highlight patterns of connectivity

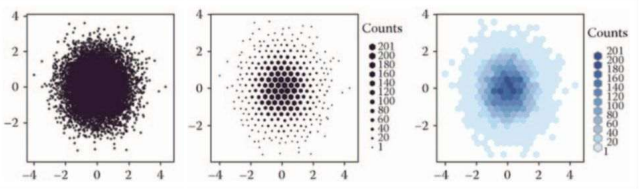
Matrix view (2)

Rows and columns sorted in the order in which data was retrieved via the Facebook API

Can you guess what's going on here?



Viz at scale? Careful!



Before you start analyzing your data

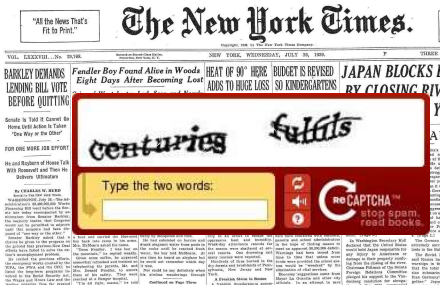
- "Do I have **missing data**?" "If data were missing, how could I know?"
- "Do I have **corrupted data**?" (May arise from measurement errors, processing bugs, etc.)
- **Parse/transform data** into appropriate format for your specific analysis (see "Part 1: Data models")
- Don't be surprised if you need to come back to this stage!

Desiderata

It's always ideal if you can put your hands on the **code/documentation about the dataset** you are analyzing (provenance)

It's always ideal if the provided **data format is nicely parseable** (otherwise you need regexes, or maybe even pay humans)

Highly non-parseable data



Entire NY Times archive (since 1851) digitized as of 2015

Feedback

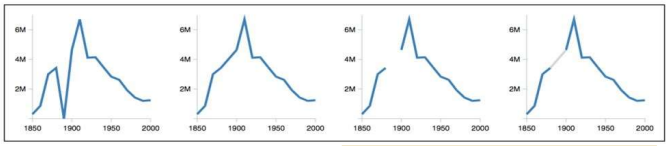
Give us feedback on this lecture here:
https://go.epfl.ch/ada2025-lec2-feedback

Feedback form available for each lecture and lab session

- What did you (not) like about this lecture?
- What was (not) well explained?
- On what would you like more (fewer) details?
- Is it nicer to follow the lecture online or offline?
- …

Dealing with missing data

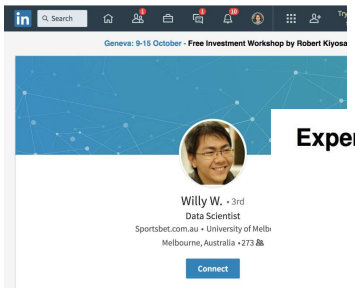
U.S. census counts of people working as "farm laborers"; values from 1890 are **missing due to records being burned in a fire**



- Set values to zero?
- Interpolate based on existing data?
- Omit missing data?

Knowledge about domain and data collection should drive your choice!

Inconsistent data: “My name is Willy”



First name	Last name
Willy	NULL
...	...

Experiments on Pattern-based Rel

Willy Yap and Timothy Baldwin
NICTA Victoria Research Laboratory
Department of Computer Science and Software E
University of Melbourne
willy@csse.unimelb.edu.au, tim@csse.unir