# Data Ingress: The Foundation of Data Science

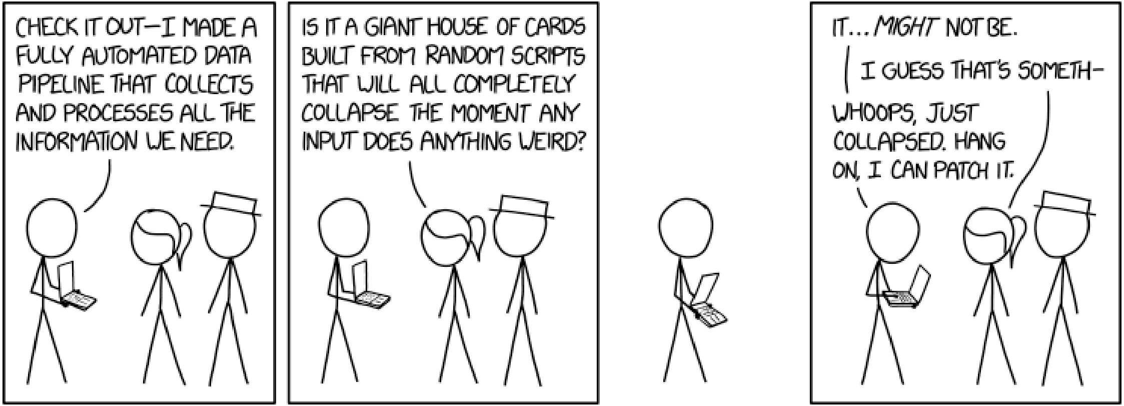
## Applied Data Science (COMS30050)

Nirav Ajmeri
Acknowledgement: Seth Bullock, Niall Twomey, Ryan McConville, Raúl Santos-Rodríguez

School of Computer Science
University of Bristol

# Agenda

In this lecture we will discuss data ingress—*crucial first step of data science*.




https://xkcd.com/2054/

# By the end of this lecture you should be...

- ... familiar with the most used data structures in data science
- ... familiar with the most used data formats in data science
- ... familiar about how to work with APIs

Later lectures will introduce...

- ... cleaning data
- ... working with databases
- ... visualising data
- ... 'data science' :-)

# Jupyter Notebooks

- Jupyter notebooks will be very useful throughout the unit
- These will demonstrate the concepts/examples directly in the browser

Useful links:

| | |
|---|---|
| Git | `https://www.atlassian.com/git/tutorials/install-git` |
| Python | `https://www.python.org/` |
| Jupyter | `https://jupyter.org/install` |
| In-browser | `https://repl.it/languages/python` |
| | `https://colab.research.google.com` |

# Outline of the lecture

Data structures

- List, Array, matrix, Dictionary

Data formats

- CSV, pandas, JSON, HDF5

Web-scraping and APIs

- Beautiful Soup, Regular expressions, Scrapy

# What is Data Ingress?

- Process of collecting and acquiring data from various sources
  - First step in the data science lifecycle
- Quality of the data ingress process directly impacts the quality of the analysis and insights
  - Garbage In, Garbage Out!

# Data structures for data science

Native data structures

| | |
|---|---|
| list | python list |
| set | python set |
| dict | python dictionary |

Data structures from other modules

| | |
|---|---|
| np.array | numpy array |
| pandas.DataFrame | pandas dataframe |

# Object persistence between sessions

## Serialisation

Serialisation is the process of translating data structures or objects from memory into a format that can be stored

## Deserialisation

Deserialisation is the inverse process; translating data structures that have been stored in a particular format to memory

---

`https://en.wikipedia.org/wiki/Serialization`

# Serialisation of data structures

- Bespoke serialisation and deserialisation methods can be crafted manually
- **Example** Define a simple serialisation format for list or array:
  - Instantiate an output file object
  - Write each element of the list to file, letting one and only one element be written per line
  - Close the file

## Serialisation

```python
# Create list
v = [1, 2, 3, 4, 5]
# Write it to file
f = open("d.ivec", "w")
for el in v:
  f.write("%d\n" % el)
f.close()








# Alternatively:
with open("d.ivec", "w") as f:
f.write("\n".join(map(str, v)))
```

## Deserialisation

```python
# Instantiate list
v = []

# Read the file
f = open("d.ivec", "r")
for l in f.readlines():
  v.append(int(l))
f.close()

# Print features of the data
print(v)
# [1, 2, 3, 4, 5]
print(len(v))    #5
print(v[2])      #3

# Alternatively:
with open("d.ivec", "r") as f:
  v = map(int, f.readlines())
```

# Problems with bespoke serialisation:

- Very specific use case
- Format not standardised
- The above example is not robust in its current (naïve) state
- Needs to be tested against many test cases
- No object metadata encoded (e.g. data type, length)
- Every data structure (e.g. matrices, dictionaries, list of strings) requires a (de)serialisation method

... but should be fine if using in well-controlled situations

# Comma-separated values (CSV)

- Very suited to tabular data, particularly matrices
- A row is stored as a line
- Each element in the row is separated by a comma
- Example CSV file:

```
1, 2, 3
4, 5, 6
7, 8, 9
```

# Loading CSV file with Python

- Easy to write own parser, but will use the pandas python package to load CSV data: `http://pandas.pydata.org/`
- The pandas library performs intelligent type conversion and checking
- Provides a powerful DataFrame object

```python
from pandas import read_csv
df = read_csv("csv.csv")
print(df)
```

```
   0  1  2
0  1  2  3
1  4  5  6
2  7  8  9
```

- CSV files can be a very time efficient and space efficient format choice for tabular data

https://tools.ietf.org/html/rfc4180

# Data Types in Data Science

Data Types

- Dense data
- Sparse data
- Text data
- Structured/relational data
- Categorical/Ordinal
- Date/time
- Lat/lon

Data Characteristics

- CSV should be fine :-)
- Don't store in dense format
- How to store efficiently
- Handle relationships
- Handling categorical constraints
- Retrieving time zone
- Retrieving location

# Serialising generic objects

JSON (JavaScript Object Notation)

- Human readable, `dict`-like format
- Very robust language; suits many purposes

HDF5 (Hierarchical Data Format)

- Binary format
- File system-like access

Will not cover other formats in this lecture (e.g. XML (and related variants), Protocol buffers or YAML and others)

https://en.wikipedia.org/wiki/JSON

https://en.wikipedia.org/wiki/Hierarchical_Data_Format

https://en.wikipedia.org/wiki/Category:Data_serialization_formats

# JavaScript Object Notation (JSON)

- JSON is a syntax for storing and exchanging data
- JSON is text, written with JavaScript object notation standard
- Although initially designed for javascript, JSON is a common serialisation in many languages, APIs, and communication frameworks, e.g. REST APIs.
- We can convert JSON into objects in memory
  - May need to create specific conversion process.
- JSON is a very well defined standard
  - "Because it is so simple, it is not expected that the JSON grammar will ever change. This gives JSON, as a foundational notation, tremendous stability"

## JSON

```json
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

## Python

```python
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212␣555-1234"
    },
    {
      "type": "mobile",
      "number": "123␣456-7890"
    }
  ],
  "children": [],
  "spouse": None
}
```

# Some distinctions between JSON and Python dicts...

| | dict | JSON |
|---|---|---|
| Missing values | None | null |
| String character | '' or "" | "" only |
| Dictionary keys | any hashable object | strings |

Demonstrations:

- JSON validation: `http://www.jsonlint.com`
- JSON files can become large (due to key repetition). Transposing lists of dictionaries into a dictionary of lists will save space in general.

## Original JSON

```json
[
  {
    "type": "home",
    "number": "212 555-1234"
  },
  {
    "type": "mobile",
    "number": "123 456-7890"
  },
  {
    "type": "home2",
    "number": "212 555-1234"
  },
  {
    "type": "mobile2",
    "number": "123 456-7890"
  }
]
```

## Transposed JSON

```json
{
  "type": [
    "home",
    "mobile",
    "home2",
    "mobile2"
  ],
  "number": [
    "212 555-1234",
    "123 456-7890",
    "212 555-1234",
    "123 456-7890"
 ]
}
```

# Hierarchical Data Format 5 (HDF5)

- HDF5 is a format for serialising data
- Core concepts:
  - Datasets: array-like collections of data
  - Groups: folder-like structures that contain datasets and other groups
  - Metadata: add information that pertains to all datasets
- HDF5 lets you store huge amounts of numerical data, and easily manipulate that data from numpy
- Thousands of datasets can be stored in a single file, categorised and tagged however you want
- Unlike numpy arrays, they support a variety of transparent storage features such as compression, error-detection, and chunked I/O

```python
import h5py
import numpy as np

# Create a HDF5 file
f = h5py.File("mytestfile.hdf5", "w")

# Add a new dataset to the file: integer array of length 100
dset = f.create_dataset("mydataset", (100,), dtype="i")

# Assign values to the dataset
dset[...] = np.arange(100)

# Add a group called subgroup, with a dataset underneath
dset2 = f.create_dataset("subgroup/dataset_two", (10,), dtype="i")

# Store metadata in the HDF5 file object
dset.attrs["author"] = "nt"
dset.attrs["date"] = "24/01/2018"
```

http://docs.h5py.org/en/latest/quick.html

Effectively, you can see HDF5 as a file system within a file, where files are datasets and folders are groups. However, the HDF Group doesn't seem to like this comparison. The major differences are as follows:

- An HDF5 file is portable: the entire structure is contained in the file and doesn't depend on the underlying file system. However it does depend on the HDF5 library.
- HDF5 datasets have a rigid structure: they are all homogeneous (hyper)rectangular numerical arrays, whereas files in a file system can be anything.
- You can add metadata to groups, whereas file systems don't support this.

http://cyrille.rossant.net/moving-away-hdf5/

# Data Collection Methods

Imagine you want to analyse customer sentiment on social media. What data collection methods could you use?

# Data Collection Methods

- Web Scraping: Extracting data from websites using automated tools
- APIs: Accessing data programmatically through APIs provided by various services
- Databases: Querying a database to extract relevant data
- Sensors: Collecting data from sensors and other devices
- Surveys and questionnaires: Gathering directly from individuals
- Publicly available datasets: Accessing datasets from government agencies, research institutions and other organisations

# Next: Web Scraping