Applied Data Science

Data Ingress

Dr. Niall Twomey

University of Bristol

Outline of the lecture

- Data structures for data science
  - List
  - Array, matrix
  - Dictionary
- Data ingress
  - CSV, pandas
  - JSON
  - HDF5
- Web-scraping and APIs
  - Beautiful Soup
  - Regular expressions
  - Scrapy

Motivation

- You cannot do applied data science if you cannot access your data
- You need to be able to produce reliable representations
- You need to select appropriate data structures to suit the contexts of the application

Data structures for data science:

- Lists (`list`)
- Numpy vectors/matrices (`np.array`)
- Set (`set`)
- Dictionaries (`dict`)

Data structures for data science:

- Lists (`list`)
- Numpy vectors/matrices (`np.array`)
- Set (`set`)
- Dictionaries (`dict`)

Examples:

- See notebooks at `https://github.com/njtwomey/ADS`
- Will interact with these data structures in Python. As I go through examples it may be useful to go through the examples with me `https://repl.it/languages/python`

Python lists and dictionaries

- Creating
- Appending
- Iterating
- Comprehension, filtering, mapping

---

https://github.com/njtwomey/ADS/blob/master/01_lists.ipynb
https://github.com/njtwomey/ADS/blob/master/02_dicts.ipynb

Serialisation

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

- Examples: images, videos, etc.
- We don't insist that we should be able to interpret serialised data
- We expect that the software that we use can interpret the formats

---

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

Serialisation

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

- Examples: images, videos, etc.
- We don't insist that we should be able to interpret serialised data
- We expect that the software that we use can interpret the formats

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 (if desired)

Serialisation of data structures:

- Bespoke serialisation and deserialisation methods can be crafted manually (if desired)

- Example: Define a simple serialisation format for list:
  - 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

```
# Create list
dd = [1, 2, 3, 4, 5]

# Write it to file
fil = open("data.int_vec", "w")
for el in dd:
    fil.write("%d\n" % el)
fil.close()
```

```
# Alternatively, save with:
# src = "\n".join(map(str, dd))
# fil.write(src)
```

Serialisation

```
# Create list
dd = [1, 2, 3, 4, 5]

# Write it to file
fil = open("data.int_vec", "w")
for el in dd:
    fil.write("%d\n" % el)
fil.close()

# Alternatively, save with:
# src = "\n".join(map(str, dd))
# fil.write(src)
```

Deserialisation

```
# Instantiate list
dd = []

# Parse the file
fil = open("data.int_vec", "r")
for line in fil.readlines():
    dd.append(int(line))
fil.close()

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

# Alternatively, load with:
# dd = map(int, fil.readlines())
```

Problems with bespoke serialisation:

- Suitable for specific, well-regulated problems
- Not tolerant to non-compliant inputs
- Doesn't solve the problem of serialising general objects
- Need to account for all edge cases (e.g. blank rows)
- Will need to write many test cases (e.g. non-integer rows)

Problems with bespoke serialisation:

- Suitable for specific, well-regulated problems
- Not tolerant to non-compliant inputs
- Doesn't solve the problem of serialising general objects
- Need to account for all edge cases (e.g. blank rows)
- Will need to write many test cases (e.g. non-integer rows)

- How would the code above generalise to matrices, for example?

Serialisation of data structures:

- Serialisation of general data structures requires the serialisation standard to be at least as expressive as the underlying data type.

[1] https://en.wikipedia.org/wiki/Comma-separated_values
[2] https://en.wikipedia.org/wiki/JSON
[3] https://en.wikipedia.org/wiki/Hierarchical_Data_Format

Serialisation of data structures:

- Serialisation of general data structures requires the serialisation standard to be at least as expressive as the underlying data type.
- CSV (Comma-separated values)[1]
  - Most suitable for tabular data

[1] https://en.wikipedia.org/wiki/Comma-separated_values
[2] https://en.wikipedia.org/wiki/JSON
[3] https://en.wikipedia.org/wiki/Hierarchical_Data_Format

Serialisation of data structures:

- Serialisation of general data structures requires the serialisation standard to be at least as expressive as the underlying data type.
- CSV (Comma-separated values)[1]
  - Most suitable for tabular data
- JSON (JavaScript Object Notation)[2]
  - Text-based serialisation
  - Human readable

[1] https://en.wikipedia.org/wiki/Comma-separated_values
[2] https://en.wikipedia.org/wiki/JSON
[3] https://en.wikipedia.org/wiki/Hierarchical_Data_Format

Serialisation of data structures:

- Serialisation of general data structures requires the serialisation standard to be at least as expressive as the underlying data type.
- CSV (Comma-separated values)[1]
  - Most suitable for tabular data
- JSON (JavaScript Object Notation)[2]
  - Text-based serialisation
  - Human readable
- HDF5 (Hierarchical Data Format)[3]
  - HDF is supported by many commercial and non-commercial software platforms

[1] https://en.wikipedia.org/wiki/Comma-separated_values
[2] https://en.wikipedia.org/wiki/JSON
[3] https://en.wikipedia.org/wiki/Hierarchical_Data_Format

Summary so far

- Familiarised ourselves with python fundamentals
- Discussed specific serialisation and deserialisation

Summary so far

- Familiarised ourselves with python fundamentals
- Discussed specific serialisation and deserialisation

- But when will the Data Science start?!

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

- Several python packages can load CSV data:
  - `https://docs.python.org/2/library/csv.html`
  - `http://pandas.pydata.org/`
- The `pandas` library performs intelligent type conversion and checking[4]
- Provides a convenient `DataFrame` object[5]

Source code

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

Output

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

[4] `https://github.com/njtwomey/ADS/blob/master/03_csv_pandas.ipynb`
[5] These objects will be covered lecture 4.

Loading CSV file with python

- Several python packages can load CSV data:
  - `https://docs.python.org/2/library/csv.html`
  - `http://pandas.pydata.org/`
- The `pandas` library performs intelligent type conversion and checking[4]
- Provides a convenient `DataFrame` object[5]

Source code

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

Output

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

- Does not cope well with non-tabular data

[4] `https://github.com/njtwomey/ADS/blob/master/03_csv_pandas.ipynb`
[5] These objects will be covered lecture 4.

JavaScript Object Notation (JSON)

- JSON is a syntax for storing and exchanging data
- JSON is text, written with JavaScript object notation standard
- We can convert JSON into objects in memory
- JSON is a very well defined standard

- Although initially designed for javascript, JSON is a common serialisation in many languages, APIs, and communication frameworks

---

`http://www.w3schools.com/js/js_json_intro.asp`

JSON code:

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

JSON code:

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

Python code:

```
{
  "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 usage: `https://github.com/njtwomey/ADS/blob/master/04_json_python.ipynb`
  - JSON validation: `http://www.jsonlint.com`
  - JSON in python: `https://repl.it/languages/python`

- JSON files can become large (due to `key` repetition)

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
dset1 = 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"] = "01/02/2017"
```

---

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

Summary:

- Discussed important data structures for data science
  - Lists and sets
  - Vectors/matrices
  - Dictionaries
- Introduced rules of thumb for selection of serialisation format:
  - CSV: When the data are tabular
  - JSON: In most cases
  - HDF5: When working with
  - Besopke: When absolutely necessary

Web scraping:

- Web scraping (web harvesting or web data extraction) is data scraping used for extracting data from websites
- Web scraping should be done only with the permission of the website's administrators. Doing otherwise may result in significant costs attributed to hosts
- Scraping technologies must be tolerant to several artifacts of real-world code
- The erroneous data makes information retrieval difficult
- Several tools have been developed to tackle this problem
- I will discuss BeautifulSoup[6] in this lecture (other libraries are also suitable, including *e.g.* `scrapy`[7])

[6] `https://www.crummy.com/software/BeautifulSoup/`
[7] `https://scrapy.org`

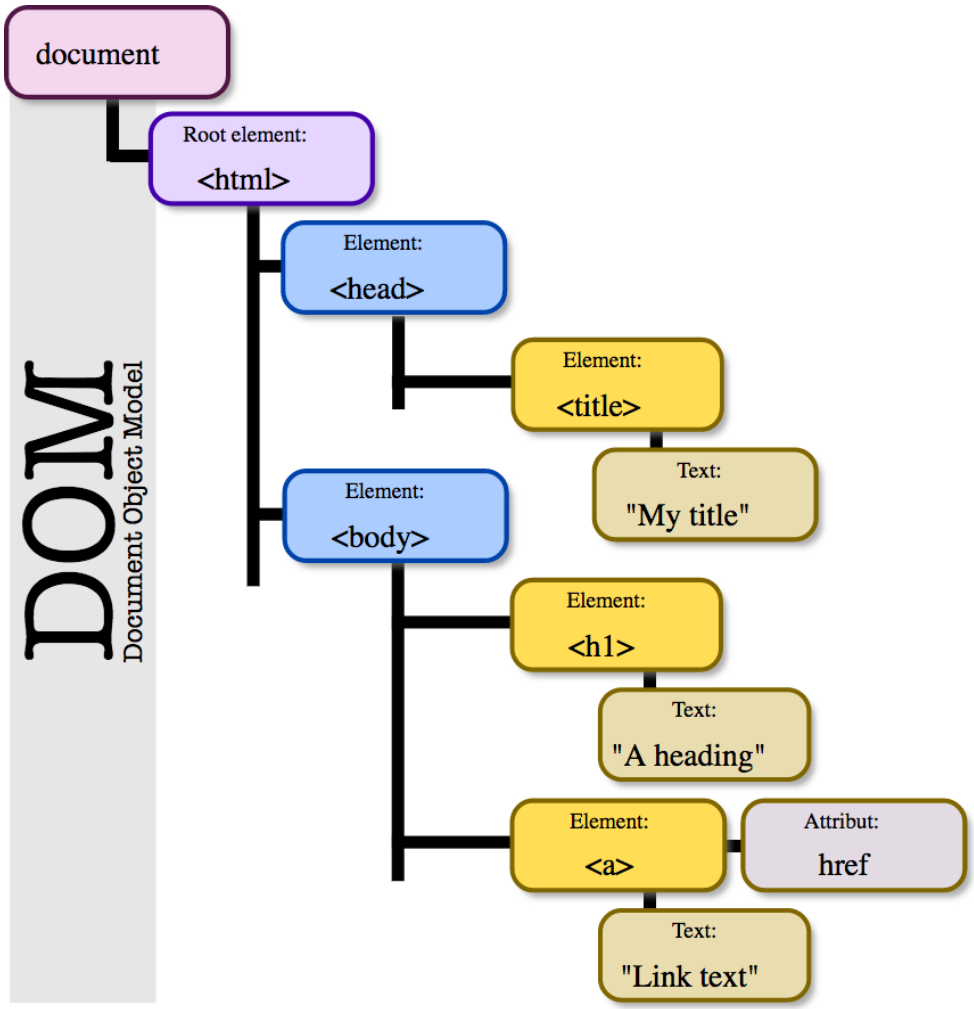
```
<!DOCTYPE html>
<html>
  <head>
    <title>Scraping</title>
  </head>
  <body class="col-sm-12">
    <h1>section1</h1>
    <p>paragraph1</p>
    <p>paragraph2</p>
    <div class="col-sm-2">
      <h2>section2</h2>
      <p>paragraph3</p>
      <p>unclosed
    </div>
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Scraping</title>
  </head>
  <body class="col-sm-12">
    <h1>section1</h1>
    <p>paragraph1</p>
    <p>paragraph2</p>
    <div class="col-sm-2">
      <h2>section2</h2>
      <p>paragraph3</p>
      <p>unclosed
    </div>
  </body>
</html>
```

- `<!DOCTYPE html>`: HTML documents must start with a type declaration.
- The HTML document is contained between `<html>` and `</html>`.
- The meta and script declaration of the HTML document is between `<head>` and `</head>`.
- The visible part of the HTML document is between `<body>` and `</body>`.
- Title headings are defined with the `<h1>` to `<h6>` tags.
- The section/division tags `<div>` are often used to segment the source.
- Paragraphs are defined with the `<p>` tag.

The DOM

- The Document Object Model (DOM) is a cross-platform and language-independent application programming interface that treats an HTML, XHTML, or XML document as a tree structure wherein each node is an object representing a part of the document.
- Tools (including BeautifulSoup) parse HTML and produce a DOM-like representation.



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

Retrieving Information

- Once the DOM has been parsed, BeautifulSoup objects may be queried
- Two query functions exist:

`find` Return the first instance in the DOM to matches the query

`find_all` Return a list of all tags that match the query

- These functions take two parameters: the first parameter defines the tag to be searched for (e.g. `head`), and the second specifies filters
- Attributes can also be handled by the search functions, e.g. to select only the segments with particular `class` attributes.

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(source, "html.parser")

print soup.find("title")
# <title>Scraping</title>

print soup.find("title").text
# Scraping

print soup.find_all("div", attrs={"class": "col-sm-2"})
# [<div class="col-sm-2">....</div>]

print soup.find_all("p")
# [<p>paragraph1</p>,
#  <p>paragraph2</p>,
#  <p>paragraph3</p>,
#  <p>unclosed\n    </p>] <= Automatically closed open tag
```

---

https://github.com/njtwomey/ADS/blob/master/05_web_scraping.ipynb

Summary of Beautiful Soup:

- Beautiful Soup is a powerful library for parsing XML/HTML documents
- With this library, one can create information extraction engines that can run autonomously
- However, BeautifulSoup suffers from serious problems:
  - Selecting the 'correct' link to follow is a strong function of the ability to identify the link of interest based only on its HTML tags. This can be difficult in poorly designed websites.
  - If the layout or formatting of information on a webpage changes, it will become necessary to reconfigure the web scraping mechanism from scratch
- Tests should be defined to detect failures
- To see an example of parsing real data with BeautifulSoup, see `https://github.com/njtwomey/ADS/blob/master/05_web_scraping.ipynb`

Web APIs:

- We have seen in the last section that parsing raw HTML is nontrivial since it will be necessary to contend with:
  - Evolving source code
  - Erroneous HTML tags

Web APIs:

- We have seen in the last section that parsing raw HTML is nontrivial since it will be necessary to contend with:
  - Evolving source code
  - Erroneous HTML tags
- Fortunately, web APIs for data retrieval exist, and these are generally less prone to the previous issues since:
  - Code is optimised for retreival and not for visual layout/aesthetics
  - Standard serialisation tools (*e.g.* JSON) are typically used
  - The core items of interest have been extracted (*e.g.* dates, URLs)

Web API examples:

- `https://github.com/toddmotto/public-apis`
- `https://en.wikipedia.org/wiki/List_of_open_APIs`

Web API examples:

- `https://github.com/toddmotto/public-apis`
- `https://en.wikipedia.org/wiki/List_of_open_APIs`
- These APIs, although similar in principle, will be very different in practise.
- However, writing code for APIs is generally simpler, requires less maintenance, and results in faster overall code

Web API examples:

- `https://github.com/toddmotto/public-apis`
- `https://en.wikipedia.org/wiki/List_of_open_APIs`
- These APIs, although similar in principle, will be very different in practise.
- However, writing code for APIs is generally simpler, requires less maintenance, and results in faster overall code

Worked example

- I will continue using ‘The Guardian’ since they have an open platform
- The task will be to acquire a list of recent posts from the ‘technology’ section of the newspaper
  `http://open-platform.theguardian.com/`
  `http://open-platform.theguardian.com/documentation/`

RESTful APIs:

- Representational state transfer (REST) or RESTful Web services are one way of providing interoperability between computer systems on the Internet
- In most circumstances API keys are required before data can be accessed

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

RESTful APIs:

- Representational state transfer (REST) or RESTful Web services are one way of providing interoperability between computer systems on the Internet
- In most circumstances API keys are required before data can be accessed

- RESTful APIs define a collection of resources

  `https://content.guardianapis.com/sections?api-key=test`

- For each resource, RESTful APIs additionally provide a list of items a

  `https://content.guardianapis.com/technology?api-key=test`

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

```python
# Specify the arguments
args = {
    "section": "technology",
    "order-by": "newest",
    "api-key": "test",
    "page-size": "100"
}

# Construct the URL
base_url = create_guardian_url(args)

# Make the request and extract the source
response = json.loads(requests.get(url).text)

# Print the data that is available
print response.keys()
#  ["currentPage", "orderBy", "pageSize", "pages", "results",
#   "startIndex", "status", "total", "userTier"]
```

https://github.com/njtwomey/ADS/blob/master/06_web_scraping_api.ipynb

Generic queries: regular expressions:

- Regular expressions are sequences of characters that define a search pattern
- In Python there are two main options foe executing regular expressions: `re.match` and `re.search`
- These two functions are similar, but with distinct differences

```
# This function attempts to match RE pattern to the whole string
re.match(pattern, string, flags=0)

# This function searches for *first* occurrence of a pattern
re.search(pattern, string, flags=0)
```

---

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

```
import re

line = "Cats are smarter than dogs"
matchObj = re.match( r"(.*) are (.*?) .*", line, re.I)

print "matchObj.group():", matchObj.group()
# "Cats are smarter than dogs"

print "matchObj.group(1):", matchObj.group(1)
# "Cats"

print "matchObj.group(2):", matchObj.group(2)
# "dogs"
```

```
def criterion(tag):
    return tag.has_attr("class") and re.search("close", tag.text)

res = soup.find_all(criterion)

print res[0]
print
print res[1]
print
```

---

http://pythex.org/

Summary on APIs:

- API-based querying is robust, reliable, well maintained and documented with a static schema
  - HTML-based web scraping is not
- Information extraction is not based on fickle naming conventions of tag attributes
- Since only content is acquired (and no images, javascript or style files) the bandwidth spent to acquire data is reduced significantly. This naturally lends itself to faster processing
- Since access is acquired through API keys, it is easy for the service provider to manage the bandwith and throughput of its service as necessary
- The data structures being received from APIs are complex, and need specific serialisation tools to be built.

Resources for today's lecture

- Python tutorials:
  `https://docs.python.org/2/tutorial/`
- 'Python for Data Analysis'
  `http://shop.oreilly.com/product/0636920023784.do`
- JSON Verification:
  `http://jsonlint.com/`
- Serialisation techniques:
  `https://en.wikipedia.org/wiki/Comparison_of_data_serialization_formats`
- Worked examples
  `https://github.com/njtwomey/ADS`