05_OSEMN

January 19, 2025

1 Data science is OSEMN

According to a popular model, the elements of data science are

- **O**btaining data
- **S**crubbing data
- **E**xploring data
- **M**odeling data
- i**N**terpreting data

and hence the acronym OSEMN, pronounced as "Awesome".

We will start with the **O**, moving towards the rest later, but first let's have a quick look at what it all boils down to

A data scientist typically is someone with expertise about data but no specific domain knowledge. Instead, if we do data analysis as scientist (physicists, in particular), we typically know what we want to achieve and we are exploiting the data in order to prove or disprove an hypotesis. The fact that we have an hypotesis leading our data analysis is a help. See for example the following case: this is population dynamics. A physicist knows about LV model and immediately understands what the correlation he/she sees is about. A data scientist would just conclude "there is a correlation" and stop their anaylisis there

*The scientist's way is the most **efficient** way: it is so easy to encounter spurious correlations in the data! The prior (or prejudice, or hypotesis) is fundamental to understand if the correlation we see is random or has meaning. (see website Spurious Correlations: https://www.tylervigen.com/spurious-correlations)*

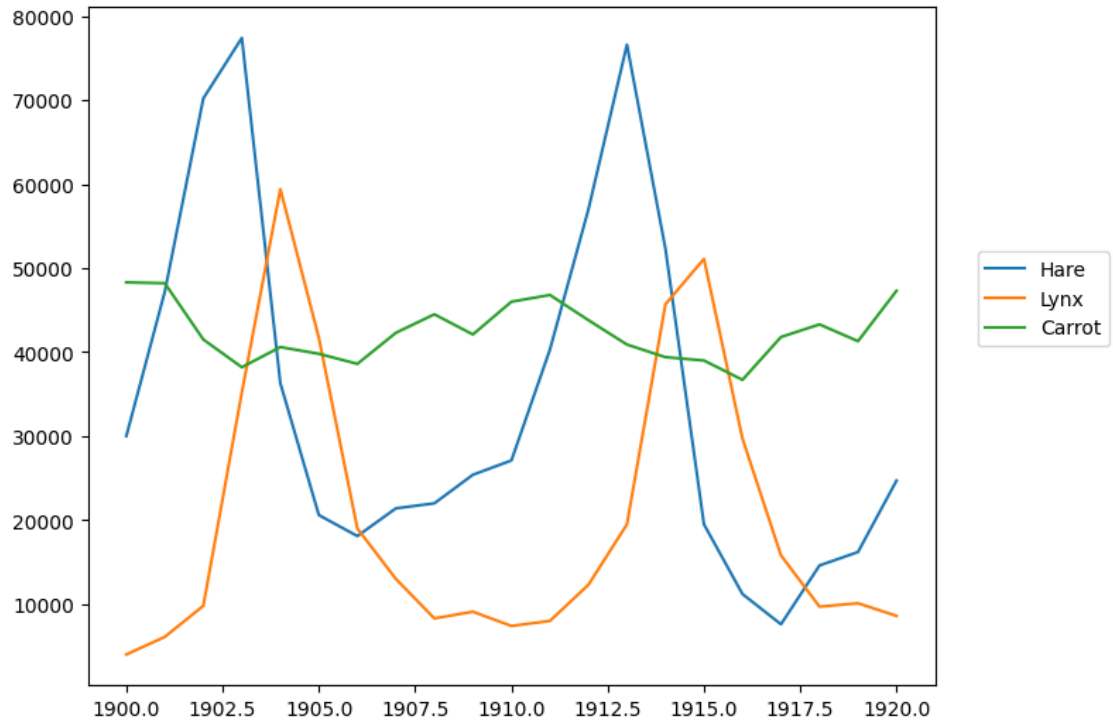
*We are in a **Physics of** Data course: lets not forget our scientific background and let's use it as an advantage. We want to do science, empowered by strong computing power, but still science.*

The process of **S**crubbing the data is very much facilitated if you have some domain knowledge about the data: e.g. you have (P,V,T) data and the temperature is T>100 celsius, of course you throw that point away.

[1]:
```
import numpy as np
data = np.loadtxt('populations.txt')
year, hares, lynxes, carrots = data.T # trick: columns to variables


from matplotlib import pyplot as plt
```

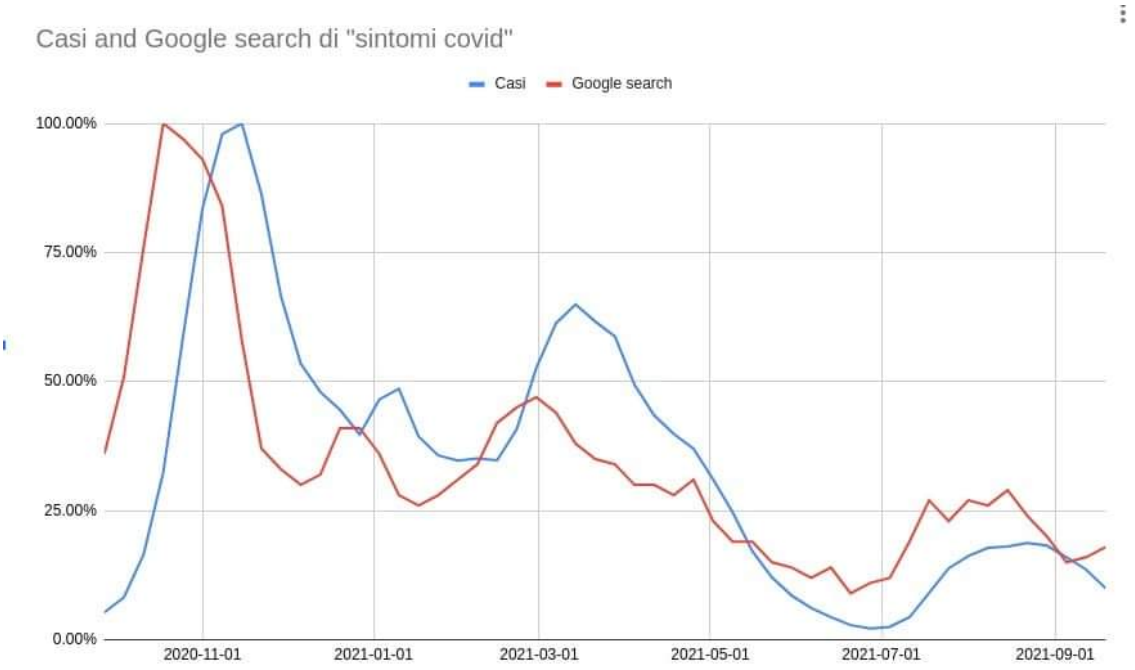
```
%matplotlib inline

plt.figure(figsize=(12,6))
plt.axes([0.2, 0.1, 0.5, 0.8])
plt.plot(year, hares, year, lynxes, year, carrots)
plt.legend(('Hare', 'Lynx', 'Carrot'), loc=(1.05, 0.5))
```

[1]: <matplotlib.legend.Legend at 0x1161905c0>



[2]:
```
from IPython.display import Image
Image("cases_vs_searches.jpeg")
```

[2]:



By plotting the data a clear (and reasonable) correlations between pray and predator becomes evident. How can it be quantified? Is that statistical significant? What about the correlation between carrots and hares? Is that evident? Is that significant?

Finding correlations in data is the main goal of data science, though that is not the end of the story: as this precious site demonstrates, **correlations is not causation**.

(I've been invited to a school of Philosophy of Science to talk about the role of ML in Physics and they even asked me to write a summary of that. You find it here, in Italian.. (humanists like that better than English))

1.0.1 Exercise:

write an algorithm that determins and quantifies a correlation between two time series. Use as an example the hare-lynx-carrot dataset.

N.B: A useful trick If in the cells below you import a packpage not yet installed, you can either install it the usual way, or run a cell like the following:

[ ]:
```
# uncomment and set  NAME_OF_THE_PACKAGE to what you need
'''
import subprocess
subprocess.call(['pip', 'install', 'NAME_OF_THE_PACKAGE'])
'''
```

2 Obtaining and processing (remote) data

Accessing data is a really serious business. Data can sit on public or on remote machines. In the case of the former, things may be straightforward, whereas in the latter case you need to worry about a few things.

In both cases, depending on the size of the dataset, the managment of the dataset can become extremely complicated. We won't deal here with large datasets, which require a whole course per se.., but still care should be put. In particular, it is not wise to keep (and even worse commit) data into a git repository!

The suggestion is then to create a directory somewhere and copy the example datasets there. From a terminal:

```
# create a data directory in your home directory
mkdir ~/data/

# check the content (it's empty now of course)
ls -ltr ~/data/

# in the case you need to move there:
cd ~/data/
```

Whatsapp was sold for 98bilion dollars. How come, if it is free on your phone? Because it has all users' informations

Keep this in mind: if you dont pay for a product, it means you are the product

Obtaining good quality data is not easy and not cheap

2.0.1 Download data from a server (url)

A nice set of interesting datasets can be found on this server that collects training/test data for machine learning developments. Several of those pertein physical sciences, it is worth browsing through those.

You can download any of those, in the following we will consider a dataset from the MAGIC experiment (astrophysics). For that we will the `wget` command

[5]:
```
# get the dataset and its description on the proper data directory
!curl -o magic04_data.csv https://archive.ics.uci.edu/ml/
 ↪machine-learning-databases/magic/magic04.data
!curl -o magic04_names.csv https://archive.ics.uci.edu/ml/
 ↪machine-learning-databases/magic/magic04.names
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 1442k    0 1442k    0     0   158k      0 --:--:--  0:00:09 --:--:--  322k
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  5400    0  5400    0     0  10738      0 --:--:-- --:--:-- --:--:-- 10735
```

[6]:
```
# print the description. This can (and better) be done from a terminal
!cat ~/data/magic04.names
```

```
cat: /Users/miriamzara/data/magic04.names: No such file or directory
```

[ ]:
```
!cat ~/data/magic04.data
```

It is possible to download and load remote files via their url's directly from within python (and thus on a jupyter session). This is a rather powerful tool as it allows http communications, IO streaming and so on.

Care should be put as the dataset is stored in memory.

[7]:
```
import urllib.request
url ='https://archive.ics.uci.edu/ml/machine-learning-databases/magic/magic04.
 ↪names'
with urllib.request.urlopen(url) as data_file:
    #print (data_file.read(300))
    for line in data_file:
        print (line)
```

```
---------------------------------------------------------------------------
SSLCertVerificationError                  Traceback (most recent call last)
File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/urllib/
 ↪request.py:1344, in AbstractHTTPHandler.do_open(self, http_class, req,␣
 ↪**http_conn_args)
   1343 try:
-> 1344     h.request(req.get_method(), req.selector, req.data, headers,
   1345               encode_chunked=req.has_header('Transfer-encoding'))
   1346 except OSError as err: # timeout error

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/http/
 ↪client.py:1319, in HTTPConnection.request(self, method, url, body, headers,␣
 ↪encode_chunked)
   1318 """Send a complete request to the server."""
-> 1319 self._send_request(method, url, body, headers, encode_chunked)

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/http/
 ↪client.py:1365, in HTTPConnection._send_request(self, method, url, body,␣
 ↪headers, encode_chunked)
   1364     body = _encode(body, 'body')
-> 1365 self.endheaders(body, encode_chunked=encode_chunked)

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/http/
 ↪client.py:1314, in HTTPConnection.endheaders(self, message_body,␣
 ↪encode_chunked)
   1313     raise CannotSendHeader()
-> 1314 self._send_output(message_body, encode_chunked=encode_chunked)
```

```
File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/http/
 ↪client.py:1074, in HTTPConnection._send_output(self, message_body,␣
 ↪encode_chunked)
   1073 del self._buffer[:]
-> 1074 self.send(msg)
   1076 if message_body is not None:
   1077
   1078     # create a consistent interface to message_body

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/http/
 ↪client.py:1018, in HTTPConnection.send(self, data)
   1017 if self.auto_open:
-> 1018     self.connect()
   1019 else:

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/http/
 ↪client.py:1460, in HTTPSConnection.connect(self)
   1458     server_hostname = self.host
-> 1460 self.sock = self._context.wrap_socket(self.sock,
   1461                                         server_hostname=server_hostname)

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/ssl.py:
 ↪455, in SSLContext.wrap_socket(self, sock, server_side,␣
 ↪do_handshake_on_connect, suppress_ragged_eofs, server_hostname, session)
    449 def wrap_socket(self, sock, server_side=False,
    450                 do_handshake_on_connect=True,
    451                 suppress_ragged_eofs=True,
    452                 server_hostname=None, session=None):
    453     # SSLSocket class handles server_hostname encoding before it calls
    454     # ctx._wrap_socket()
--> 455     return self.sslsocket_class._create(
    456         sock=sock,
    457         server_side=server_side,
    458         do_handshake_on_connect=do_handshake_on_connect,
    459         suppress_ragged_eofs=suppress_ragged_eofs,
    460         server_hostname=server_hostname,
    461         context=self,
    462         session=session
    463     )

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/ssl.py:
 ↪1046, in SSLSocket._create(cls, sock, server_side, do_handshake_on_connect,␣
 ↪suppress_ragged_eofs, server_hostname, context, session)
   1045             raise ValueError("do_handshake_on_connect should not be␣
 ↪specified for non-blocking sockets")
-> 1046         self.do_handshake()
   1047 except (OSError, ValueError):
```

```
File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/ssl.py:
 ↪1317, in SSLSocket.do_handshake(self, block)
   1316         self.settimeout(None)
-> 1317     self._sslobj.do_handshake()
   1318 finally:

SSLCertVerificationError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify␣
 ↪failed: self-signed certificate in certificate chain (_ssl.c:1000)

During handling of the above exception, another exception occurred:

URLError                                  Traceback (most recent call last)
Cell In[7], line 3
      1 import urllib.request
      2 url ='https://archive.ics.uci.edu/ml/machine-learning-databases/magic/
 ↪magic04.names'
----> 3 with urllib.request.urlopen(url) as data_file:
      4     #print (data_file.read(300))
      5     for line in data_file:
      6         print (line)

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/urllib/
 ↪request.py:215, in urlopen(url, data, timeout, cafile, capath, cadefault,␣
 ↪context)
    213 else:
    214     opener = _opener
--> 215 return opener.open(url, data, timeout)

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/urllib/
 ↪request.py:515, in OpenerDirector.open(self, fullurl, data, timeout)
    512     req = meth(req)
    514 sys.audit('urllib.Request', req.full_url, req.data, req.headers, req.
 ↪get_method())
--> 515 response = self._open(req, data)
    517 # post-process response
    518 meth_name = protocol+"_response"

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/urllib/
 ↪request.py:532, in OpenerDirector._open(self, req, data)
    529     return result
    531 protocol = req.type
--> 532 result = self._call_chain(self.handle_open, protocol, protocol +
    533                           '_open', req)
    534 if result:
    535     return result
```

```
File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/urllib/
 ↪request.py:492, in OpenerDirector._call_chain(self, chain, kind, meth_name,␣
 ↪*args)
    490 for handler in handlers:
    491     func = getattr(handler, meth_name)
--> 492     result = func(*args)
    493     if result is not None:
    494         return result

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/urllib/
 ↪request.py:1392, in HTTPSHandler.https_open(self, req)
   1391 def https_open(self, req):
-> 1392     return self.do_open(http.client.HTTPSConnection, req,
   1393                         context=self._context)

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/urllib/
 ↪request.py:1347, in AbstractHTTPHandler.do_open(self, http_class, req,␣
 ↪**http_conn_args)
   1344         h.request(req.get_method(), req.selector, req.data, headers,
   1345                   encode_chunked=req.has_header('Transfer-encoding'))
   1346     except OSError as err: # timeout error
-> 1347         raise URLError(err)
   1348     r = h.getresponse()
   1349 except:

URLError: <urlopen error [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify␣
 ↪failed: self-signed certificate in certificate chain (_ssl.c:1000)>
```

2.0.2 Copy data from a remote machine

Often datasets are not available on websites but rather they are sitting on some remote machine. Several tools are there that can allow you to get hold off remote data, even from within python (e.g. paramiko), but best in this case is to get a local copy. E.g. from a terminal:

```
scp guest104@gatep.fisica.unipd.it:~/data/data_000637.* ~/data/
```

by issuing that command you are immediately exposed to the most relevant problem in obtaining the data: **permissions/authorization**.

Indeed that will not work (as you don't have an account on that machine and I'd be put into jail if I gave you the password), still you'll need that file later, so "wget" it:

[ ]:
```
!wget https://www.dropbox.com/s/69xe1d5f19nvtw3/data_000637.dat -P ~/data/

# copy the interpreted version as well
!wget https://www.dropbox.com/s/xvjzaxzz3ysphme/data_000637.txt -P ~/data/
```

Secondily (essentially a further consequence of the same issue), the remote machine itself may have accessibility restrictions, e.g. being behind a firewall. In that case you may need to use a tunnel:

```
ssh -L 1234:<address of R known to G>:22 <user at G>@<address of G>
```

```
scp -P 1234 <user at R>@127.0.0.1:/path/to/file file-name-to-be-copied
```

In summary, just getting the data is a complicated business.

2.0.3 Working on data stored remotely without copying it

If the data is huge in size, you dont want to copy it on your machine. This needs and infastructure (i.e. a batch system) or interactive analysis with Jupyter, where the backend is distributed (data is stored on several computers). We will see it on later courses, but keep the thing in mind.

3 Data Formats

Datasets can be stored in a gazillion different ways: oftentimes, their format is tailored to the application they are used for. The results is a mess, but luckily more and more standards are being established. **Python has "readers" for most of the formats, another reason for being the optimal programming language for data analysis**. In the following we revise some of the most used formats:

- **.text**
- **.csv**
- **.bin, .hex** (binary or exadecimal)
- **.json**
- **.hdf5**
- **.root**

3.0.1 Text files

Plain text files are commonly used for "readibility", at the price of a very poor storing efficiency due to their low entropy. UTF-8 is the most common **encoding**

The encoding is the set of rules for the conversion binary <-> alphabetic character: at the end, binary digits are the content of every digital file, the difference is just how you interpret it as symbols.

Reading (and writing) text files in python is straightforward:

[ ]:
```
file_name = "/Users/mzanetti/data/magic04.data"

# mode can be specified for writing, reading or both
with open(file_name, mode='r') as f:
    # print-out the whole file
    # print (f.read())
    for line in f:
        ## print line by line
        print (line)
        ## each line is a string, you need to split it yourself
        for c in line.split(): print(c) # check the functionalities of the␣
 ↪split() method
```

3.0.2 CSV files

If you are lucky, text files are already framed into a defined structured, in a "table-like" manner. These files are called "**c**omma **s**eparated **v**alues" (csv), even though the separator may well not be the "," symbol.

Python has packages to deal with that. More often than not, csv files have comments (e.g. starting with '#'), which cannot be interpreted by the reader. Tricks like:

```
csv.reader(row for row in f if not row.startswith('#'))
```

may be useful

[10]:
```
import csv


with open('magic04_data.csv') as data_file:
    for line in csv.reader(data_file, delimiter=','):
        # the delimiter is often guessed by the reader
        # again note that elements of each line are treated as strings
        # if you need to convert them into numbers, you need to to that yourself
        # like it is done here:
        ##
        fLength,fWidth,fSize,\
        fConc,fConc1,fAsym,\
        fM3Long,fM3Trans,fAlpha,fDist = map(float,line[:-1]) #conversion to␣
 ↪float
        ##
        category = line[-1]
        print␣
 ↪(fLength,fWidth,fSize,fConc,fConc1,fAsym,fM3Long,fM3Trans,fAlpha,fDist,␣
 ↪end=' , ')
        print (category)
        break
```

```
28.7967 16.0021 2.6449 0.3918 0.1982 27.7004 22.011 -8.2027 40.092 81.8828 , g
```

3.0.3 Binary (hexadecimal) files

The output of sensors often is stored as hexadecimal files. Information is packed in a well defined format (similarly to how floating point numbers are formatted). To read and process hexadecimal files in python you need to use the `b` option of `open` and progress along the file at step of *defined length* (depending on the size of the words information is packed into)

There are several tool to display and edit hex/bin files, e.g. this one

The following is an example from data collected from an FPGA implementing a TDC (**T**ime to **D**igital **C**onverter). Relevant infomation are the coordinates of the TDC channels and their time measurements.

[ ]:
```
import struct, time

with  open('/Users/mzanetti/data/data_000637.dat','rb') as file:
    file_content=file.read()
    word_counter=0
    word_size = 8 # size of the word in bytes
    for i in range(0, len(file_content), word_size):
        word_counter+=1
        if word_counter>100: break
        time.sleep(0.1)
        thisInt = struct.unpack('<q', file_content[i:i+word_size])[0]
        head = (thisInt >> 62) & 0x3
        if head == 1:
            fpga     = (thisInt >> 58) & 0xF
            tdc_chan = (thisInt >> 49) & 0x1FF
            orb_cnt  = (thisInt >> 17) & 0xFFFFFFFF
            bx       = (thisInt >> 5 ) & 0xFFF
            tdc_meas = (thisInt >> 0 ) & 0x1F
            if i==0 : print ('{0},{1},{2},{3},{4},{5}'.format('HEAD', 'FPGA',␣
 ↪'TDC_CHANNEL', 'ORB_CNT', 'BX', 'TDC_MEAS'))
            print ('{0},{1},{2},{3},{4},{5}'.format(head, fpga, tdc_chan,␣
 ↪orb_cnt, bx, tdc_meas))
        else:
            print ('ERROR! head =', head)
```

3.0.4 JSON files

JSON is JavaScript Object Notation - a format used widely for web-based resource sharing. It is very similar in structure to a Python nested dictionary. Here is an example from http://json.org/example

[ ]:
```
%%file example.json
{
    "glossary": {
        "title": "example glossary",
            "GlossDiv": {
            "title": "S",
                    "GlossList": {
                "GlossEntry": {
                    "ID": "SGML",
                                    "SortAs": "SGML",
                                    "GlossTerm": "Standard Generalized Markup␣
 ↪Language",
                                    "Acronym": "SGML",
                                    "Abbrev": "ISO 8879:1986",
                                    "GlossDef": {
                        "para": "A meta-markup language, used to create markup␣
 ↪languages such as DocBook.",
```

```
                                        "GlossSeeAlso": ["GML", "XML"]
                    },
                                    "GlossSee": "markup"
                }
            }
        }
    }
}
```

[ ]:
```
!cat example.json
```

[1]:
```
import json
data = json.load(open('example.json'))
print (data)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[1], line 2
      1 import json
----> 2 data = json.load(open('example.json'))
      3 print (data)

File ~/LaboratoryOfComputationalPhysics_Y7/myenv/lib/python3.12/site-packages/
 ↪IPython/core/interactiveshell.py:324, in _modified_open(file, *args, **kwargs
    317 if file in {0, 1, 2}:
    318     raise ValueError(
    319         f"IPython won't let you open fd={file} by default "
    320         "as it is likely to crash IPython. If you know what you are␣
 ↪doing, "
    321         "you can use builtins' open."
    322     )
--> 324 return io_open(file, *args, **kwargs)

FileNotFoundError: [Errno 2] No such file or directory: 'example.json'
```

[ ]:
```
# and can be parsed using standard key lookups
data['glossary']['GlossDiv']#['GlossList']
```

JSON structure looks very much alike a dictionary, thus dumping a dict into a json file is straightforward

[ ]:
```
data = {
    "name": "Alice",
    "age": 25,
    "skills": ["Python", "JavaScript"]
}
```

```
# Write JSON data to a file
with open('data.json', 'w') as file:
    json.dump(data, file, indent=4)
```

3.0.5 HDF5

The HDF5 format is a versatile file format designed for storing and managing large amounts of data. HDF5 stands for **H**ierarchical **D**ata **F**ormat version **5** and is widely used in fields like scientific computing, machine learning, and big data applications due to its efficiency and scalability.

The main concepts associated with HDF5 are

- Hierarchical Structure: files are organized in a tree-like structure, similar to a file system, with groups (like folders) and datasets (like files). This structure allows for logically organizing complex data relationships.
- Efficient Storage: Optimized for storing large datasets, including multidimensional arrays, and allows efficient I/O operations. It supports compression to save storage space.
- Self-Describing: The file contains metadata that describes the data, such as the dimensions, data type, and attributes of datasets. This makes it easier to understand the file content without external documentation.

Structure of an HDF5 File: * Groups: Like directories, groups can contain other groups or datasets. * Datasets: These are arrays of data, analogous to files in a directory. * Attributes: Metadata attached to groups or datasets, like key-value pairs. / (Root Group) /group1 (Group) /dataset1 (Dataset) /dataset2 (Dataset) /subgroup (Group) /dataset3 (Dataset) /group2 (Group) /attributes (Attributes attached to a group or dataset)

let's create an hdf5 file and read it

[ ]:
```
import numpy as np
import h5py


#Now mock up some simple dummy data to save to our file.
d1 = np.random.random(size = (1000,20))
d2 = np.random.random(size = (1000,200))

print (d1.shape, d2.shape)


hf = h5py.File('data.h5', 'w')
hf.create_dataset('dataset_1', data=d1)
hf.create_dataset('dataset_2', data=d2)
hf.close()
```

```
(1000, 20) (1000, 200)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[3], line 12
```

```
      7 d2 = np.random.random(size = (1000,200))
      9 print (d1.shape, d2.shape)
---> 12 hf = h5py.File('05_LAb_OSEMN/data.h5', 'w')
     13 hf.create_dataset('dataset_1', data=d1)
     14 hf.create_dataset('dataset_2', data=d2)

File ~/LaboratoryOfComputationalPhysics_Y7/myenv/lib/python3.12/site-packages/
 ↪h5py/_hl/files.py:561, in File.__init__(self, name, mode, driver, libver,␣
 ↪userblock_size, swmr, rdcc_nslots, rdcc_nbytes, rdcc_w0, track_order,␣
 ↪fs_strategy, fs_persist, fs_threshold, fs_page_size, page_buf_size,␣
 ↪min_meta_keep, min_raw_keep, locking, alignment_threshold, alignment_interval ␣
 ↪meta_block_size, **kwds)
    552     fapl = make_fapl(driver, libver, rdcc_nslots, rdcc_nbytes, rdcc_w0,
    553                      locking, page_buf_size, min_meta_keep, min_raw_kee ,
    554                      alignment_threshold=alignment_threshold,
    555                      alignment_interval=alignment_interval,
    556                      meta_block_size=meta_block_size,
    557                      **kwds)
    558     fcpl = make_fcpl(track_order=track_order, fs_strategy=fs_strategy,
    559                      fs_persist=fs_persist, fs_threshold=fs_threshold,
    560                      fs_page_size=fs_page_size)
--> 561     fid = make_fid(name, mode, userblock_size, fapl, fcpl, swmr=swmr)
    563 if isinstance(libver, tuple):
    564     self._libver = libver

File ~/LaboratoryOfComputationalPhysics_Y7/myenv/lib/python3.12/site-packages/
 ↪h5py/_hl/files.py:241, in make_fid(name, mode, userblock_size, fapl, fcpl,␣
 ↪swmr)
    239     fid = h5f.create(name, h5f.ACC_EXCL, fapl=fapl, fcpl=fcpl)
    240 elif mode == 'w':
--> 241     fid = h5f.create(name, h5f.ACC_TRUNC, fapl=fapl, fcpl=fcpl)
    242 elif mode == 'a':
    243     # Open in append mode (read/write).
    244     # If that fails, create a new file only if it won't clobber an
    245     # existing one (ACC_EXCL)
    246     try:

File h5py/_objects.pyx:54, in h5py._objects.with_phil.wrapper()

File h5py/_objects.pyx:55, in h5py._objects.with_phil.wrapper()

File h5py/h5f.pyx:122, in h5py.h5f.create()

FileNotFoundError: [Errno 2] Unable to create file (unable to open file: name =
 ↪'05_LAb_OSEMN/data.h5', errno = 2, error message = 'No such file or␣
 ↪directory', flags = 13, o_flags = 602)
```

[4]:
```
hf = h5py.File('data.h5', 'r')

print (hf.keys())

n1 = hf.get('dataset_1')
print ("n1", n1)

n1 = np.array(n1)
print (n1.shape)
```

```
<KeysViewHDF5 ['dataset_1', 'dataset_2']>
n1 <HDF5 dataset "dataset_1": shape (1000, 20), type "<f8">
(1000, 20)
```

here is another example

[ ]:
```
import h5py
import numpy as np
import os

# creating a HDF5 file
import datetime
if not os.path.exists('example.hdf5'):

    with h5py.File('example.hdf5','w') as f:
        project = f.create_group('project') #main group
        expt1 = project.create_group('expt1') # sub-groups
        expt2 = project.create_group('expt2') # sub-groups
        expt1.create_dataset('counts', (100,), dtype='i')
        expt2.create_dataset('values', (1000,), dtype='f')

        expt1['counts'][:] = range(100)
        expt2['values'][:] = np.random.random(1000)

# reading it
with h5py.File('example.hdf5') as f:
    project = f['project']
    print (project['expt1']['counts'][:10])
    print (project['expt2']['values'][:10])
```

3.1 Final mentions : Pandas and ROOT

3.1.1 Pandas

Pandas is not actually a data format. It is a tool for reading, processing and even directly visualizing formatted datasets, and it is actually the *most convenient* one. We are going to dedicate a whole lecture to it. Below, see just a couple examples.

```
[ ]: import pandas as pd
     file_name="/Users/mzanetti/data/data_000637.txt"
     data=pd.read_csv(file_name,nrows=10,skiprows=range(1,1))
     data
```

```
[ ]: !cat /Users/mzanetti/data/data_000637.txt
```

```
[ ]: import pandas as pd
     file_name="/Users/mzanetti/data/magic04.data"
     data=pd.read_csv(file_name,nrows=1000)
     data.columns=['fLength','fWidth','fSize',
             'fConc','fConc1','fAsym',
             'fM3Long','fM3Trans','fAlpha','fDist','category']
     data
```

```
[ ]: %matplotlib inline

     data.plot.scatter("fLength","fWidth",)
```

```
[ ]: # HISTOGRAMS directly from Pandas
     data.hist("fAlpha")
```

3.1.2 ROOT

ROOT needs a special mention. It is still nowadays, and by far, the most convenient tool to store and manage complex datasets pertaining physics experiments where "events" are recorded, in particular High Energy, Nuclear, Astro physics. It allows a nested structure, with complex data objects (classes) and references between them.

ROOT per se is obnoxious as it has been developed in the years as a way-too-many-purposes package, but **its I/O is still formidable**.

Installing ROOT is (or at leaset used to be) a pain. (bare) ROOT files can be opened with non-ROOT library, uproot (check its git repo). A data structure, RDataFrame similar to the ones developed for modern Data Science applications has been put in production.

Bottomiline: Root is still useful to analyze data which cannot be stored in a table with fixed-named columns. This because data is stored as vectors of different lengths, and even objects (e.g. an entire particle track) can be stored as variables. For all the other purposes, do not use root. An even when you need to use root data, you better use it from libraries like uproot and not natively.

Professor Zanetti is very upset that root came out this way and Pandas was developed by non-physicist. It is a shame that physicists came out with a tool like root, they could have done much better, he says.

```
[7]: import uproot


     events = uproot.open("https://scikit-hep.org/uproot3/examples/Zmumu.
      ↪root")["events"]
```

```
events
```

```
---------------------------------------------------------------------------
SSLCertVerificationError                  Traceback (most recent call last)
File ~/LaboratoryOfComputationalPhysics_Y7/myenv/lib/python3.12/site-packages/
 ↪aiohttp/connector.py:1116, in TCPConnector._wrap_create_connection(self,␣
 ↪addr_infos, req, timeout, client_error, *args, **kwargs)
   1109         sock = await aiohappyeyeballs.start_connection(
   1110             addr_infos=addr_infos,
   1111             local_addr_infos=self._local_addr_infos,
   (…)
   1114             loop=self._loop,
   1115         )
-> 1116         return await self._loop.create_connection(*args, **kwargs,␣
 ↪sock=sock)
   1117 except cert_errors as exc:

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/asyncio/
 ↪base_events.py:1126, in BaseEventLoop.create_connection(self,␣
 ↪protocol_factory, host, port, ssl, family, proto, flags, sock, local_addr,␣
 ↪server_hostname, ssl_handshake_timeout, ssl_shutdown_timeout,␣
 ↪happy_eyeballs_delay, interleave, all_errors)
   1123         raise ValueError(
   1124             f'A Stream Socket was expected, got {sock!r}')
-> 1126 transport, protocol = await self._create_connection_transport(
   1127     sock, protocol_factory, ssl, server_hostname,
   1128     ssl_handshake_timeout=ssl_handshake_timeout,
   1129     ssl_shutdown_timeout=ssl_shutdown_timeout)
   1130 if self._debug:
   1131     # Get the socket from the transport because SSL transport closes
   1132     # the old socket and creates a new SSL socket

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/asyncio/
 ↪base_events.py:1159, in BaseEventLoop._create_connection_transport(self, sock␣
 ↪protocol_factory, ssl, server_hostname, server_side, ssl_handshake_timeout,␣
 ↪ssl_shutdown_timeout)
   1158 try:
-> 1159     await waiter
   1160 except:

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/asyncio/
 ↪sslproto.py:575, in SSLProtocol._on_handshake_complete(self, handshake_exc)
    574 else:
--> 575     raise handshake_exc
    577 peercert = sslobj.getpeercert()

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/asyncio/
 ↪sslproto.py:557, in SSLProtocol._do_handshake(self)
    556 try:
```

```
--> 557     self._sslobj.do_handshake()
    558 except SSLAgainErrors:

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/ssl.py:
 ↪917, in SSLObject.do_handshake(self)
    916 """Start the SSL/TLS handshake."""
--> 917 self._sslobj.do_handshake()

SSLCertVerificationError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify␣
 ↪failed: unable to get local issuer certificate (_ssl.c:1000)

The above exception was the direct cause of the following exception:

ClientConnectorCertificateError           Traceback (most recent call last)
File ~/LaboratoryOfComputationalPhysics_Y7/myenv/lib/python3.12/site-packages/
 ↪fsspec/implementations/http.py:423, in HTTPFileSystem._info(self, url,␣
 ↪**kwargs)
    421 try:
    422     info.update(
--> 423         await _file_info(
    424             self.encode_url(url),
    425             size_policy=policy,
    426             session=session,
    427             **self.kwargs,
    428             **kwargs,
    429         )
    430     )
    431     if info.get("size") is not None:

File ~/LaboratoryOfComputationalPhysics_Y7/myenv/lib/python3.12/site-packages/
 ↪fsspec/implementations/http.py:833, in _file_info(url, session, size_policy,␣
 ↪**kwargs)
    832 elif size_policy == "get":
--> 833     r = await session.get(url, allow_redirects=ar, **kwargs)
    834 else:

File ~/LaboratoryOfComputationalPhysics_Y7/myenv/lib/python3.12/site-packages/
 ↪aiohttp/client.py:696, in ClientSession._request(self, method, str_or_url,␣
 ↪params, data, json, cookies, headers, skip_auto_headers, auth,␣
 ↪allow_redirects, max_redirects, compress, chunked, expect100,␣
 ↪raise_for_status, read_until_eof, proxy, proxy_auth, timeout, verify_ssl,␣
 ↪fingerprint, ssl_context, ssl, server_hostname, proxy_headers,␣
 ↪trace_request_ctx, read_bufsize, auto_decompress, max_line_size,␣
 ↪max_field_size)
    695 try:
--> 696     conn = await self._connector.connect(
    697         req, traces=traces, timeout=real_timeout
    698     )
    699 except asyncio.TimeoutError as exc:
```

```
File ~/LaboratoryOfComputationalPhysics_Y7/myenv/lib/python3.12/site-packages/
 ↪aiohttp/connector.py:544, in BaseConnector.connect(self, req, traces, timeout
    543         await trace.send_connection_create_start()
--> 544 proto = await self._create_connection(req, traces, timeout)
    545 if traces:

File ~/LaboratoryOfComputationalPhysics_Y7/myenv/lib/python3.12/site-packages/
 ↪aiohttp/connector.py:1050, in TCPConnector._create_connection(self, req,␣
 ↪traces, timeout)
   1049 else:
-> 1050     _, proto = await self._create_direct_connection(req, traces, timeou )
   1052 return proto

File ~/LaboratoryOfComputationalPhysics_Y7/myenv/lib/python3.12/site-packages/
 ↪aiohttp/connector.py:1394, in TCPConnector._create_direct_connection(self,␣
 ↪req, traces, timeout, client_error)
   1393 assert last_exc is not None
-> 1394 raise last_exc

File ~/LaboratoryOfComputationalPhysics_Y7/myenv/lib/python3.12/site-packages/
 ↪aiohttp/connector.py:1363, in TCPConnector._create_direct_connection(self,␣
 ↪req, traces, timeout, client_error)
   1362 try:
-> 1363     transp, proto = await self._wrap_create_connection(
   1364         self._factory,
   1365         timeout=timeout,
   1366         ssl=sslcontext,
   1367         addr_infos=addr_infos,
   1368         server_hostname=server_hostname,
   1369         req=req,
   1370         client_error=client_error,
   1371     )
   1372 except (ClientConnectorError, asyncio.TimeoutError) as exc:

File ~/LaboratoryOfComputationalPhysics_Y7/myenv/lib/python3.12/site-packages/
 ↪aiohttp/connector.py:1118, in TCPConnector._wrap_create_connection(self,␣
 ↪addr_infos, req, timeout, client_error, *args, **kwargs)
   1117 except cert_errors as exc:
-> 1118     raise ClientConnectorCertificateError(req.connection_key, exc) from
 ↪exc
   1119 except ssl_errors as exc:

ClientConnectorCertificateError: Cannot connect to host scikit-hep.org:443 ssl:
 ↪True [SSLCertVerificationError: (1, '[SSL: CERTIFICATE_VERIFY_FAILED]␣
 ↪certificate verify failed: unable to get local issuer certificate (_ssl.c:
 ↪1000)')]

The above exception was the direct cause of the following exception:
```

```
FileNotFoundError                         Traceback (most recent call last)
Cell In[7], line 4
      1 import uproot
----> 4 events =␣
 ↪uproot.open("https://scikit-hep.org/uproot3/examples/Zmumu.root")["events"]
      5 events

File ~/LaboratoryOfComputationalPhysics_Y7/myenv/lib/python3.12/site-packages/
 ↪uproot/reading.py:142, in open(path, object_cache, array_cache,␣
 ↪custom_classes, decompression_executor, interpretation_executor, **options)
    133 if not isinstance(file_path, str) and not (
    134     hasattr(file_path, "read") and hasattr(file_path, "seek")
    135 ):
    136     raise ValueError(
    137         "'path' must be a string, pathlib.Path, an object with 'read'␣
 ↪and "
    138         "'seek' methods, or a length-1 dict of {file_path: object_path} ␣
 ↪"
    139         f"not {path!r}"
    140     )
--> 142 file = ReadOnlyFile(
    143     file_path,
    144     object_cache=object_cache,
    145     array_cache=array_cache,
    146     custom_classes=custom_classes,
    147     decompression_executor=decompression_executor,
    148     interpretation_executor=interpretation_executor,
    149     **options,
    150 )
    152 if object_path is None:
    153     return file.root_directory

File ~/LaboratoryOfComputationalPhysics_Y7/myenv/lib/python3.12/site-packages/
 ↪uproot/reading.py:561, in ReadOnlyFile.__init__(self, file_path, object_cache␣
 ↪array_cache, custom_classes, decompression_executor, interpretation_executor,␣
 ↪**options)
    556 self.hook_before_create_source()
    558 source_cls, file_path = uproot._util.file_path_to_source_class(
    559     file_path, self._options
    560 )
--> 561 self._source = source_cls(file_path, **self._options)
    563 self.hook_before_get_chunks()
    565 if self._options["begin_chunk_size"] < _file_header_fields_big.size:

File ~/LaboratoryOfComputationalPhysics_Y7/myenv/lib/python3.12/site-packages/
 ↪uproot/source/fsspec.py:45, in FSSpecSource.__init__(self, file_path,␣
 ↪coalesce_config, **options)
     41 self._async_impl = self._fs.async_impl
     43 self._file = None
```

```
---> 45 self._open()
     47 self.__enter__()

File ~/LaboratoryOfComputationalPhysics_Y7/myenv/lib/python3.12/site-packages/
 ↪uproot/source/fsspec.py:59, in FSSpecSource._open(self)
     57 def _open(self):
     58     self._executor = FSSpecLoopExecutor()
---> 59     self._file = self._fs.open(self._file_path)

File ~/LaboratoryOfComputationalPhysics_Y7/myenv/lib/python3.12/site-packages/
 ↪fsspec/spec.py:1301, in AbstractFileSystem.open(self, path, mode, block_size,
 ↪cache_options, compression, **kwargs)
   1299 else:
   1300     ac = kwargs.pop("autocommit", not self._intrans)
-> 1301     f = self._open(
   1302         path,
   1303         mode=mode,
   1304         block_size=block_size,
   1305         autocommit=ac,
   1306         cache_options=cache_options,
   1307         **kwargs,
   1308     )
   1309     if compression is not None:
   1310         from fsspec.compression import compr

File ~/LaboratoryOfComputationalPhysics_Y7/myenv/lib/python3.12/site-packages/
 ↪fsspec/implementations/http.py:362, in HTTPFileSystem._open(self, path, mode,
 ↪block_size, autocommit, cache_type, cache_options, size, **kwargs)
    360 kw.update(kwargs)
    361 info = {}
--> 362 size = size or info.update(self.info(path, **kwargs)) or info["size"]
    363 session = sync(self.loop, self.set_session)
    364 if block_size and size and info.get("partial", True):

File ~/LaboratoryOfComputationalPhysics_Y7/myenv/lib/python3.12/site-packages/
 ↪fsspec/asyn.py:118, in sync_wrapper.<locals>.wrapper(*args, **kwargs)
    115 @functools.wraps(func)
    116 def wrapper(*args, **kwargs):
    117     self = obj or args[0]
--> 118     return sync(self.loop, func, *args, **kwargs)

File ~/LaboratoryOfComputationalPhysics_Y7/myenv/lib/python3.12/site-packages/
 ↪fsspec/asyn.py:103, in sync(loop, func, timeout, *args, **kwargs)
    101     raise FSTimeoutError from return_result
    102 elif isinstance(return_result, BaseException):
--> 103     raise return_result
    104 else:
    105     return return_result
```

```
File ~/LaboratoryOfComputationalPhysics_Y7/myenv/lib/python3.12/site-packages/
 ↪fsspec/asyn.py:56, in _runner(event, coro, result, timeout)
     54     coro = asyncio.wait_for(coro, timeout=timeout)
     55 try:
---> 56     result[0] = await coro
     57 except Exception as ex:
     58     result[0] = ex

File ~/LaboratoryOfComputationalPhysics_Y7/myenv/lib/python3.12/site-packages/
 ↪fsspec/implementations/http.py:436, in HTTPFileSystem._info(self, url,␣
 ↪**kwargs)
    433     except Exception as exc:
    434         if policy == "get":
    435             # If get failed, then raise a FileNotFoundError
--> 436             raise FileNotFoundError(url) from exc
    437         logger.debug("", exc_info=exc)
    439 return {"name": url, "size": None, **info, "type": "file"}

FileNotFoundError: https://scikit-hep.org/uproot3/examples/Zmumu.root
```

[ ]:
```
events.show()
```

[ ]:
```
array = events["E1"].array(library="np")
array
plt.hist(array)
```

For more complicated examples, you can take a look at the CMS experiment open data; sure you can find the Higgs boson in there, extracting the signal is way easier than obtaining, storing, and interpreting the data..