Introduction to the Outreach and Education Open Data Specifics

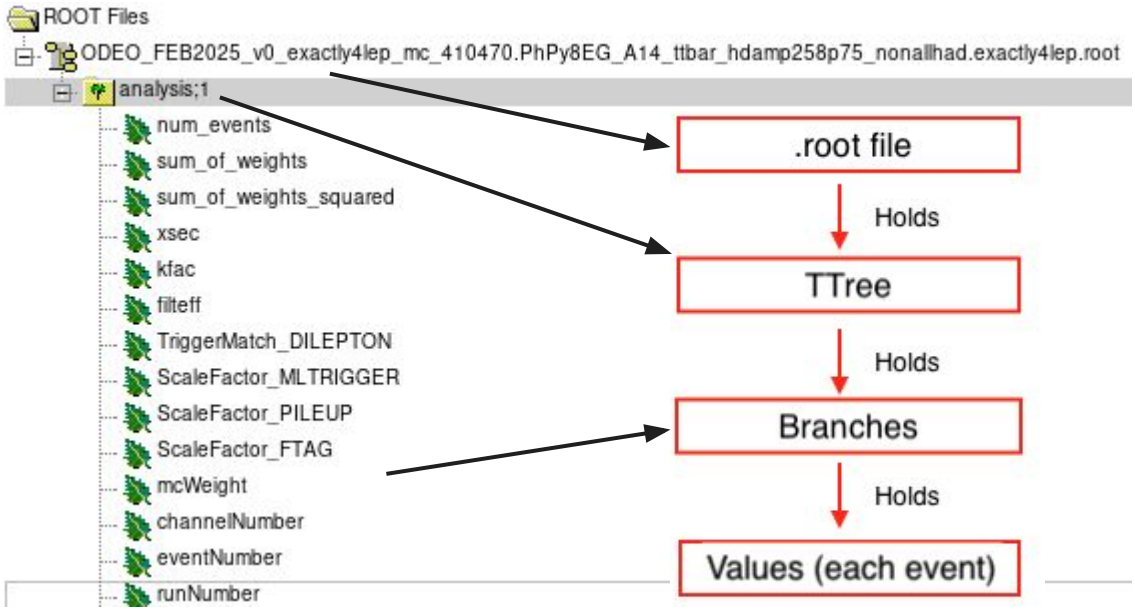
ATLAS Open Data Tutorial

24 November 2025

Eirik Gramstad | University of Oslo,
António Jacques Costa | University of Manchester

What is the format of the Open Data for E&O

- The data is stored as ROOT NTuples
  - ROOT is an open-source data analysis framework used in high energy physics (more on ROOT in Thursday's session on Open Data for research)
  - The structure of the NTuple files is made of a series of **branches**, each corresponding to a **specific variable**



To be used with ROOT...

- in either C++ or python (pyROOT)

... or without ROOT

- using the Uproot package with awkward arrays

What is stored in the ntuples?

- Electrons and muons
  - Combined into leptons
- Photons
- Tau leptons
- Small-radius jets
- Large-radius jets
- Missing transverse energy
- Scale factors to account for small differences between simulations and data
- Scale factors to scale the simulated events to the right integrated luminosity

Electron (e)	Muon (μ)	Photon (γ)
InDet & EMCAL rec.	InDet & MS rec.	InDet & EMCAL rec.
$p_T > 7$ GeV	$p_T > 7$ GeV	$E_T > 25$ GeV
$\|\eta\| < 2.47$	$\|\eta\| < 2.5$	$\|\eta\| < 2.37$

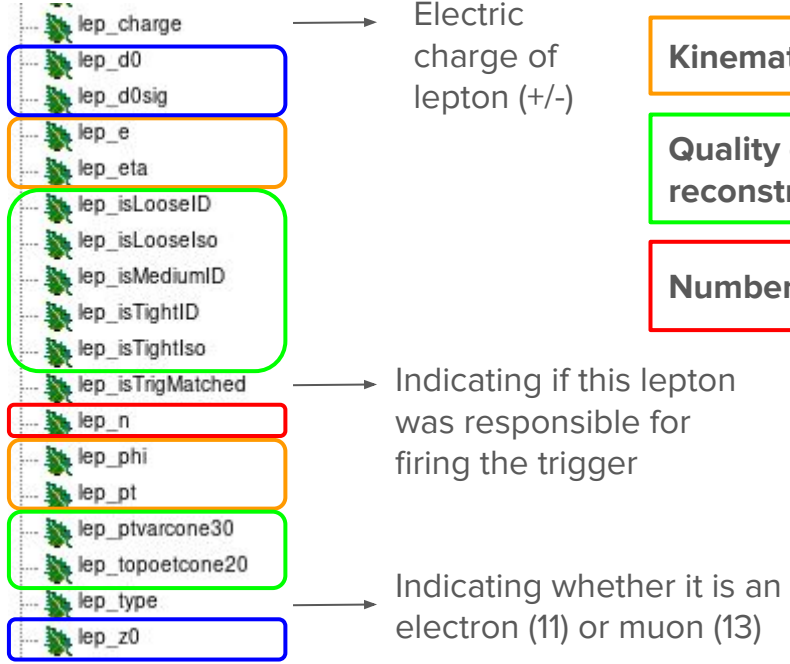
τ-leptons	Small-R jets	Large-R jets
InDet & EMCAL rec.	EMCAL & HCAL rec.	EMCAL & HCAL rec.
$p_T > 20$ GeV	anti-kt, R = 0.4	anti-kt, R = 1.0
$\|\eta\| < 2.5$	$p_T > 20$ GeV	$p_T > 250$ GeV
	$\|\eta\| < 2.5$	$\|\eta\| < 2.0$
	b-tagging (DL1dv01)	trimming: $R_{sub} = 0.2, f_{cut} = 0.05$

Everything about the content of the ntuples on the Open Data web pages

What is stored in the ntuples

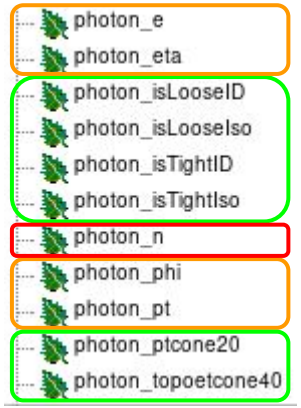
- Electrons and muons
  - Combined into a common lep-vector

Basic selection of electrons and muons in E&O release	
Electron (e)	**Muon (μ)**
InDet & EMCAL rec.	InDet & MS rec.
$p_T > 7$ GeV	$p_T > 7$ GeV
$\|\eta\| < 2.47$	$\|\eta\| < 2.5$



What is stored in the ntuples

- Photons



Kinematics

Quality of the reconstruction

Number of objects

Basic selection of photons in E&O release
InDet & EMCAL rec.
$E_T > 25$ GeV
$\|\eta\| < 2.37$

What is stored in the ntuples

- Tau leptons

How the tau decayed

Electric charge of tau lepton (+/-)

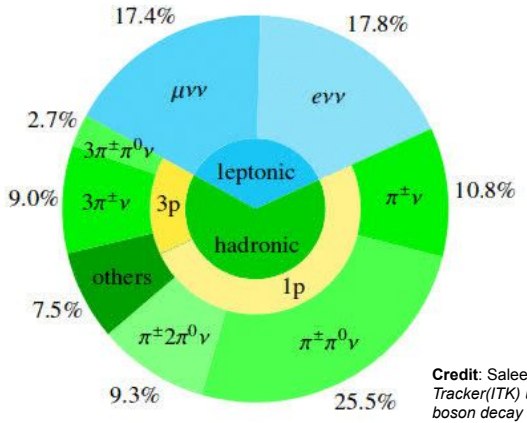
Number of tracks in hadronic tau decays

Kinematics

Quality of the reconstruction

Number of objects

Basic selection of taus in E&O release
InDet & EMCAL rec.
$p_T > 20$ GeV
$\|\eta\| < 2.5$

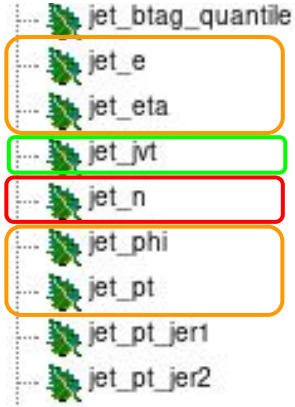


Credit: Saleem, Tasneem. (2019). *Development of pixel detector for ATLAS Inner Tracker(ITK) upgrade at HL-LHC and Searching for the Standard Model Higgs boson decay into b-quark pair with ATLAS experiment.*

What is stored in the ntuples

- Small-radius jets

Basic selection of jets in E&O release
EMCAL & HCAL rec.
anti-kt, R = 0.4
$p_T > 20$ GeV
$\|\eta\| < 2.5$
b-tagging (DL1dv01)



How likely the jet is to originate from a b-quark (next slide)

How likely the jet is to come from the primary interaction point

How the transverse momentum of the jet changes wrt. changes in the jet energy scale

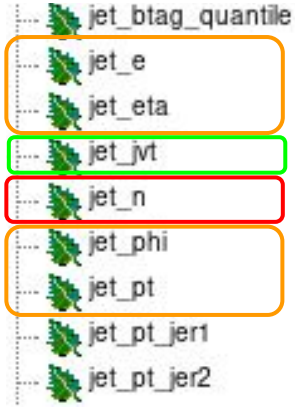
Kinematics

Quality of the reconstruction

Number of objects

What is stored in the ntuples

- *b*-jets
  - Jets passing certain criteria to be identified to originate from a *b*-quark
  - The criteria are based on a deep learning algorithm using the *DL1dv01* tagger



How likely the jet is to originate from a b-quark

How likely the jet is to come from the primary interaction point

How the transverse momentum of the jet changes wrt. to changes in the jet energy scale

Kinematics

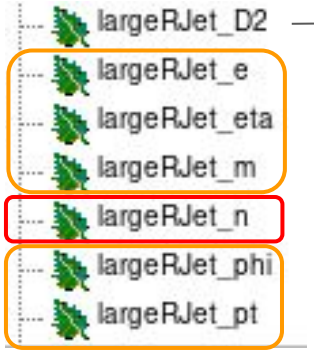
Quality of the reconstruction

Number of objects

100%	`jet_btag_quantile>=1`
85%	`jet_btag_quantile>=2`
77%	`jet_btag_quantile>=3`
70%	`jet_btag_quantile>=4`
60%	`jet_btag_quantile>=5`

What is stored in the ntuples

- Large-radius jets



Variable to identify jets from hadronic Z-boson decays (link)

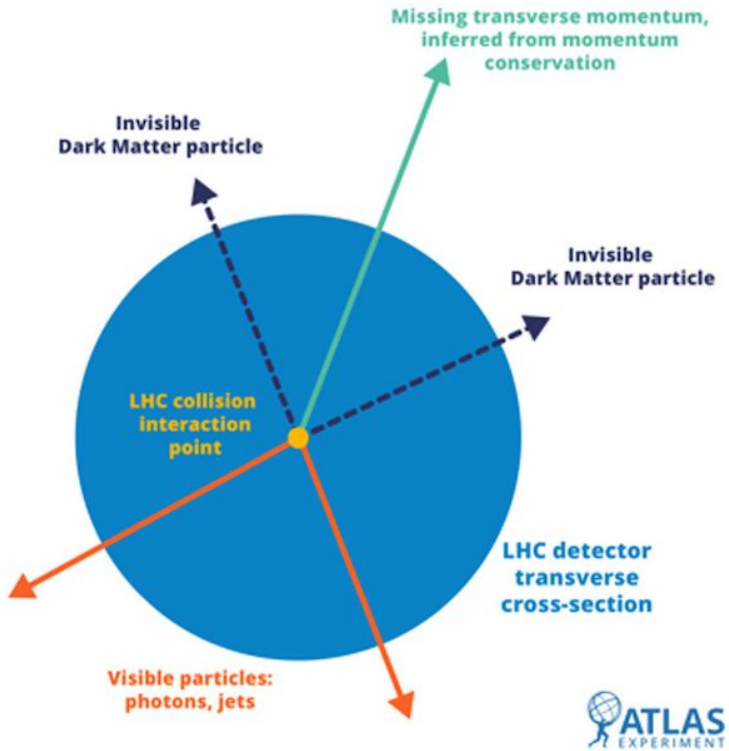
Kinematics

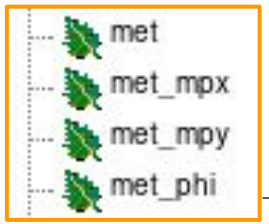
Number of objects

Basic selection of large-radius jets in E&O release
EMCAL & HCAL rec.
anti-kt, R = 1.0
$p_T > 250$ GeV
$\|\eta\| < 2.0$
trimming: $R_{sub} = 0.2$, $f_{cut} = 0.05$

What is stored in the ntuples

- Missing transverse energy (MET)
  - The incoming particles in the collision carry **negligible momentum transverse to the beamline**
  - We know that the **vector sum of the final state particles should have a transverse component that is zero**
  - After we reconstruct all of the visible particles in the event, the **missing transverse energy (MET) is calculated as the negative vector sum** of those objects
  - MET can originate from
    - Inefficiencies in our detector (“fake” MET),
    - Known invisible particles (neutrinos)
    - Unknown invisible particles (dark matter??)

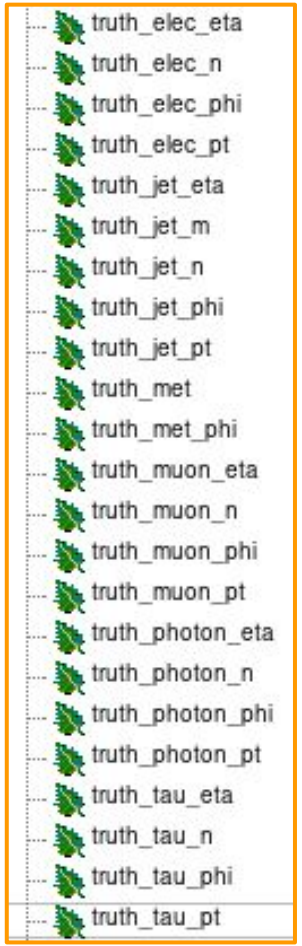




→ Information only in the plane transverse (xy) to the beam direction

What is stored in the ntuples

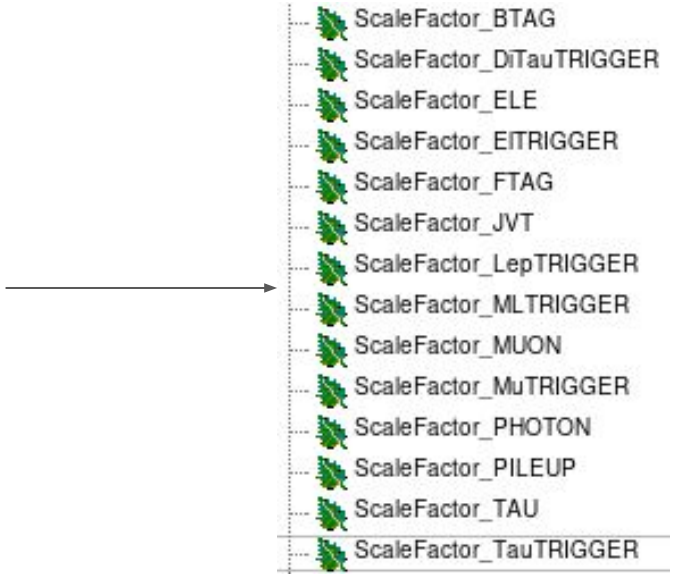
- Truth collections
  - For each of the reconstructed objects (**electrons**, **muons**, **photons**, **taus**, **missing transverse energy** and **jets**) the true objects from the generation step are stored
  - For each true object the pT, η, φ, energy/mass are stored
  - Can be matched to the reconstructed object by checking that the difference in R between the reconstructed and true object is below some threshold (typically ~0.2)



What is stored in the ntuples

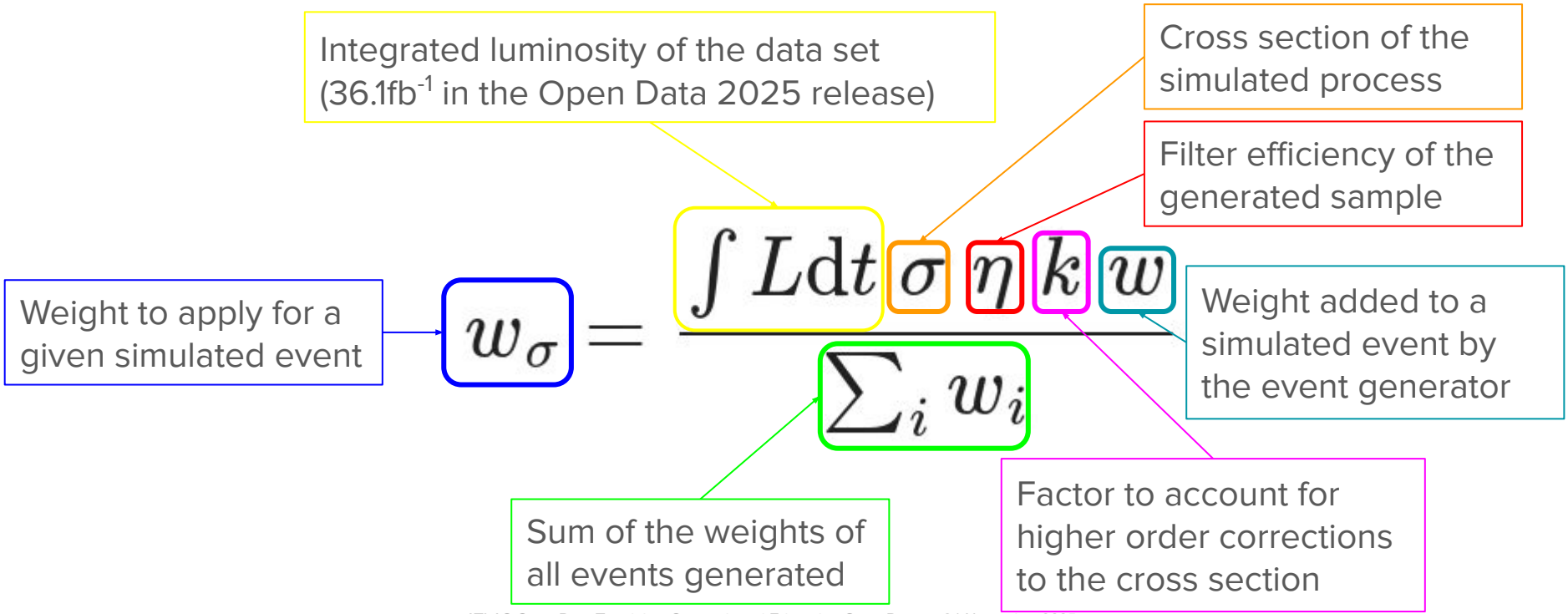
- Scale factors
  - To account for the differences in reconstruction and identification efficiencies between data and simulations we apply scale factors on the simulations
  - Also need to account for differences related to differences between the simulated and true pileup distribution (since simulations are usually made in advance of the data taking period)

Depending on the objects you use you need to apply the corresponding scale factors



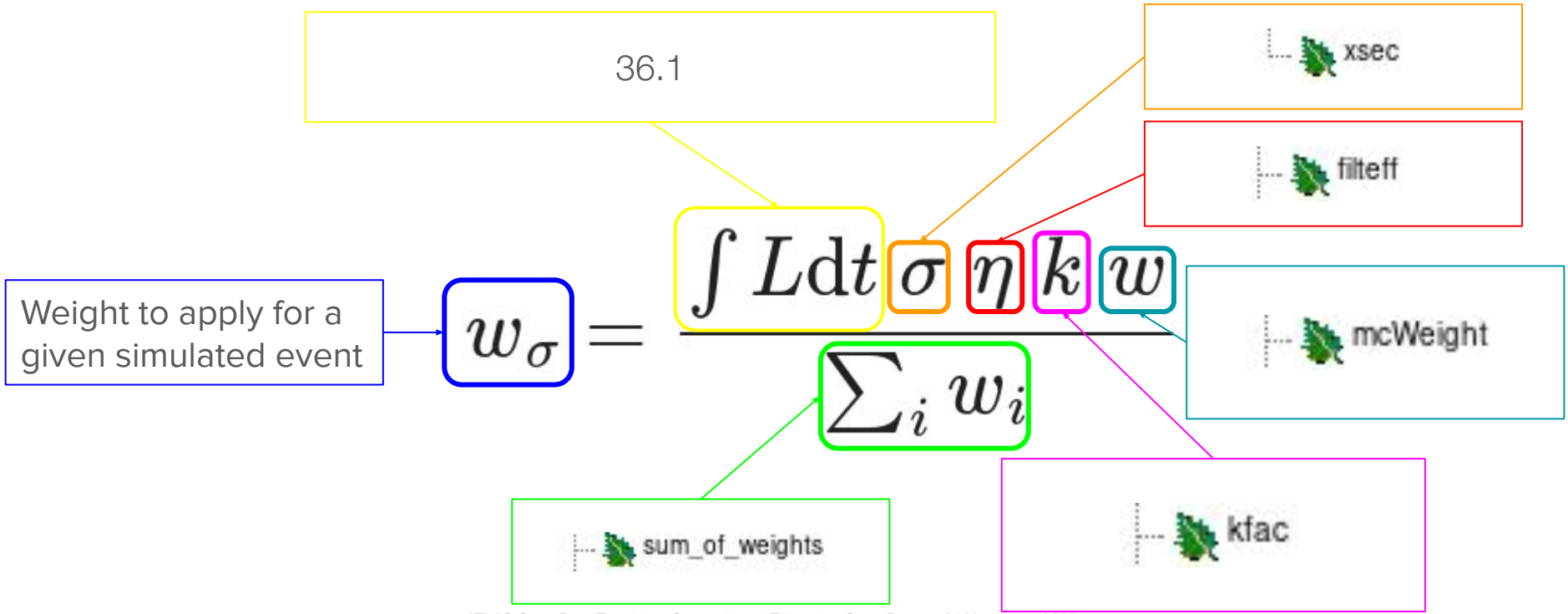
What is stored in the ntuples

- Scaling simulations to the integrated luminosity of the data set
  - You learnt in the previous lecture that the simulations are divided into many data sets representing one production (and decay mode) - each with a given cross section



What is stored in the ntuples

- Scaling simulations to the integrated luminosity of the data set
  - You learnt in the previous lecture that the simulations are divided into many data sets representing one production (and decay mode) - each with a given cross section



Which samples to use?

Selection	Collection Name
At least one lepton with at least 7 GeV of p_T and 30 GeV of missing transverse momentum (i.e. a leptonically-decaying W-boson enhanced selection)	1LMET30
Two to four leptons with at least 7 GeV of p_T each	2to4lep
At least two muons with at least 10 GeV of p_T (i.e. a leptonically-decaying Z-boson enhanced selection)	2muons
At least three jets with at least 20 GeV of p_T, at least one lepton passing tight identification requirements with at least 7 GeV of p_T, and 30 GeV of missing transverse momentum (i.e. a semi-leptonic top-quark enhanced selection)	3J1LMET30
At least two photons with at least 25 GeV of p_T each (i.e. a Higgs boson decaying to two photons enhanced selection)	GamGam
At least two jets with at least 20 GeV of p_T, at least two leptons passing tight identification requirements with at least 7 GeV of p_T, and 30 GeV of missing transverse momentum (i.e. a di-leptonic top-quark enhanced selection)	2J2LMET30
At least two jets with at least 20 GeV of p_T identified as containing at least one heavy flavor hadron using the 70% working point (i.e. a Higgs boson decaying to b-quarks enhanced selection)	2bjets
At least three leptons with at least 7 GeV of p_T each	3lep
Exactly three leptons with at least 7 GeV of p_T (i.e. a leptonically-decaying W+Z boson enhanced selection)	exactly3lep
At least four leptons with at least 7 GeV of p_T each	4lep
Exactly four leptons with at least 7 GeV of p_T (i.e. a leptonically-decaying ZZ boson or Higgs to four leptons enhanced selection)	exactly4lep

- The Open Data ntuples for E&O are published with different selections (OD E&O 13TeV 2025 details)
  - Tailored towards different types of analyses
  - A set without any particular selection applied is also published as *noskim*
    - **NOTE**! These are very large and makes your analysis considerably slower!

The names used to identify the data sets in the Open Data portal / *atlasopenmagic*

Also known as a "**skim**"

Size of some selected skimmed samples and noskim

- A lot to save by using a *skim* tailored to your analysis
- Less disk space and quicker analysis
- Will look at how you can create your own *skimmed* samples from the *noskim*

Name	Selection	Size [GB] MC	Data	Total
1LMET30	$\geq$ 1 lepton w/loose ID and $E_T^{miss} > 30$	170	176	346
2to4lep	$\geq$ 2 lepton and $\leq$ 4 leptons	55	99	154
2muons	$\geq$ 2 muons w/$p_T > 10$ GeV	22	25	46
3J1LMET30	$\geq$ 1 lepton w/tight ID and $\geq$ 3 jets and $E_T^{miss} > 30$ GeV	25	7	31
GamGam	$\geq$ 2 photons w/$p_T > 20$ GeV	7	21	28
2J2LMET30	$\geq$ 2 lepton w/tight ID and $\geq$ 2 jets and $E_T^{miss} > 30$ GeV	12	5	16
exactly2bjets	== 2 b-jets	1.5	5.4	6.8
exactly3lep	== 3 leptons w/$p_T > 10$ GeV	1.9	2.3	4.1
exactly4lep	== 4 leptons	0.68	0.73	1.4
Total		295	341	637

Name	Selection	Size [GB] MC	Data	Total
Noskim	-	509	801	1300

Demo time! (link)

DEMO TIME!

Tabular Data

- A large proportion of the data we analyse is in tabular form
  - Table consists of rows and columns
    - Each row is an individual record
    - *Example* in ML, *event* in particle physics
    - Rows are usually indexed
  - Each column is a given property of the data
    - *Feature* in ML, branch/variable in ROOT
    - Columns are usually named
  - The intersection between a row and column is a cell and usually (but not always - see later) contains a single value
- Variation: multiple entries per cell (*ragged data*) ➜ much more on this later...

Index	A	B	C	D	E	F	G
1	#	#	#	#	#	#	#
2	#	#	#	#	#	#	#
3	#	#	#	#	#	#	#
4	#	#	#	#	#	#	#
5	#	#	#	#	#	#	#
6	#	#	#	#	#	#	#
7	#	#	#	#	#	#	#
8	#	#	#	#	#	#	#

Ragged tabular Data

- Structure is still tabular but
  - Each cell can contain more than one (or no) entries
  - The number of entries in a given column varies at random from row to row
  - Each cell is an array
- This data structure is very common in particle physics
  - Relatively uncommon in data science and is not easily accommodated in the usual data science tools such as **Pandas**, **numpy**
  - This has led to development of special tools such as **ROOT**, **AwkwardArray** etc.
  - Can pose a particular problem for machine learning (more on this later...)

Index	A	B	C
0	[#,#,#]	[#,#]	[#]
1	[]	[#,#,#]	[#,#,#]
2	[#,#,#]	[#,#,#,#,#]	[#,#,#,#]
3	[#,#]	[#]	[#,#]
4	[]	[#,#,#]	[]
5	[#,#]	[#,#,#]	[#,#]
6	[#,#,#]	[#]	[#,#,#]
7	[#,#,#,#,#]	[#,#,#]	[#]

Data analysis tools

- *AwkwardArray* and uproot: “ragged NumPy”
  - *Numpy* and *Pandas* do not easily accommodate ragged data
  - This has been solved with *AwkwardArray* which allows ragged data to be handled with Numpy-like idioms
  - The *Uproot* package allows *ROOT* data structures to be converted to *Numpy* and *Pandas* structures
  - *AwkwardArray* can read data either via uproot or from the ROOT dataframe
  - We will use *AwkwardArray*, *Numpy*/*Pandas* during this tutorial (and we will look at ROOT later in the week)

- You already did the initial setup of *atlasopenmagic* in the previous demo
  - So let's just run quickly through the **first four cells**
- We will look at **data** and **one MC sample**
  - In order to not load too many events we use the 4lep skim
  - The MC samples contains simulated events of two Z-bosons, each decaying to a pair of leptons

[54]
✓ 0s

```
# We will use a skim (pre-selection) of the data so that we start from events
# with at least four leptons (electrons/muon)
skim = '4lep'
# And now we can get the files we want; note that we use "cache=True" to copy
# the files locally rather than streaming them
defs = {
    r'Data':{'dids':['data']},
    r'ZZ to 4L':{'dids': [700600], 'color': "#6b59d3" }, # purple
}

samples   = atom.build_dataset(defs, skim=skim, protocol='https', cache=True)
```

- You already did the initial setup of *atlasopenmagic* in the previous demo
  - So let's just run quickly through the **first four cells**
- We will look at **data** and **one MC sample**
  - In order to not load too many events we use the 4lep skim
  - The MC samples contains simulated events of two Z-bosons, each decaying to a pair of leptons

```
[6]
✓ 0s
    samples

{'Data': {'list': ['simplecache::https://opendata.cern.ch/eos/opendata/atlas/rucio/opendata/ODEO_FEB2025_v0_4lep_data15_periodD.4lep.root',
   'simplecache::https://opendata.cern.ch/eos/opendata/atlas/rucio/opendata/ODEO_FEB2025_v0_4lep_data15_periodE.4lep.root',
   'simplecache::https://opendata.cern.ch/eos/opendata/atlas/rucio/opendata/ODEO_FEB2025_v0_4lep_data15_periodF.4lep.root',
   'simplecache::https://opendata.cern.ch/eos/opendata/atlas/rucio/opendata/ODEO_FEB2025_v0_4lep_data15_periodG.4lep.root',
   'simplecache::https://opendata.cern.ch/eos/opendata/atlas/rucio/opendata/ODEO_FEB2025_v0_4lep_data15_periodH.4lep.root',
   'simplecache::https://opendata.cern.ch/eos/opendata/atlas/rucio/opendata/ODEO_FEB2025_v0_4lep_data15_periodJ.4lep.root',
   'simplecache::https://opendata.cern.ch/eos/opendata/atlas/rucio/opendata/ODEO_FEB2025_v0_4lep_data16_periodA.4lep.root',
   'simplecache::https://opendata.cern.ch/eos/opendata/atlas/rucio/opendata/ODEO_FEB2025_v0_4lep_data16_periodB.4lep.root',
   'simplecache::https://opendata.cern.ch/eos/opendata/atlas/rucio/opendata/ODEO_FEB2025_v0_4lep_data16_periodC.4lep.root',
   'simplecache::https://opendata.cern.ch/eos/opendata/atlas/rucio/opendata/ODEO_FEB2025_v0_4lep_data16_periodD.4lep.root',
   'simplecache::https://opendata.cern.ch/eos/opendata/atlas/rucio/opendata/ODEO_FEB2025_v0_4lep_data16_periodE.4lep.root',
   'simplecache::https://opendata.cern.ch/eos/opendata/atlas/rucio/opendata/ODEO_FEB2025_v0_4lep_data16_periodF.4lep.root',
   'simplecache::https://opendata.cern.ch/eos/opendata/atlas/rucio/opendata/ODEO_FEB2025_v0_4lep_data16_periodG.4lep.root',
   'simplecache::https://opendata.cern.ch/eos/opendata/atlas/rucio/opendata/ODEO_FEB2025_v0_4lep_data16_periodI.4lep.root',
   'simplecache::https://opendata.cern.ch/eos/opendata/atlas/rucio/opendata/ODEO_FEB2025_v0_4lep_data16_periodK.4lep.root',
   'simplecache::https://opendata.cern.ch/eos/opendata/atlas/rucio/opendata/ODEO_FEB2025_v0_4lep_data16_periodL.4lep.root']},
  'ZZ to 4L': {'list': ['simplecache::https://opendata.cern.ch/eos/opendata/atlas/rucio/opendata/ODEO_FEB2025_v0_4lep_mc_700600.Sh_2212_llll.4lep.root'],
   'color': '#6b59d3'}}
```

- We need to import several python packages for plotting, data handling and other useful things

```
[5]
✓ 3s
import numpy as np # for numerical calculations such as histogramming
import matplotlib.pyplot as plt # for plotting
import matplotlib_inline # to edit the inline plot format
#matplotlib_inline.backend_inline.set_matplotlib_formats('pdf', 'svg') # to make plots in pdf (vector) format
from matplotlib.ticker import AutoMinorLocator # for minor ticks
import uproot # for reading .root files
import awkward as ak # to represent nested data in columnar format
import vector # for 4-momentum calculations
import time # for printing time stamps
import requests # for file gathering, if needed
```

- Load the TTree in the ROOT NTuple using *Uproot*

Look at the first file in the list

Name of the TTree

```
[55]
✓ 0s
    # We open an MC data file with simulated diboson events (data set ID 700600)
    background_diboson_path = samples['ZZ to 4L']["list"][0]
    # Accessing the file from the online database
    tree_bkg = uproot.open(background_diboson_path + ":analysis")
    # We open a data file from period D in 2015 (first entry in the list)
    data_path = samples['Data']["list"][0]
    # Accessing the file from the online database
    tree_data = uproot.open(data_path + ":analysis")
```

- Let's look at the content

```
[47]
✓ 0s
print("%-30s %-50s"%("Variable Name","Size/dim/type"))
for var in sorted(tree_bkg.keys()):
    print("%-30s %-50s"%(var,tree_bkg[var].array().type))

Variable Name                  Size/dim/type
ScaleFactor_BTAG               11458 * float32
ScaleFactor_DiTauTRIGGER       11458 * float32
ScaleFactor_ELE                11458 * float32
ScaleFactor_ElTRIGGER          11458 * float32
ScaleFactor_FTAG               11458 * float32
ScaleFactor_JVT                11458 * float32
ScaleFactor_LepTRIGGER         11458 * float32
ScaleFactor_MLTRIGGER          11458 * float32
ScaleFactor_MUON               11458 * float32
ScaleFactor_MuTRIGGER          11458 * float32
ScaleFactor_PHOTON             11458 * float32
ScaleFactor_PILEUP             11458 * float32
ScaleFactor_TAU                11458 * float32
ScaleFactor_TauTRIGGER         11458 * float32
TriggerMatch_DILEPTON          11458 * float32
channelNumber                  11458 * uint32
eventNumber                    11458 * uint64
filteff                        11458 * float64
jet_btag_quantile              11458 * var * int32
jet_e                          11458 * var * float32
jet_eta                        11458 * var * float32
jet_jvt                        11458 * var * bool
jet_n                          11458 * int32
```

... + many more

Variable name

Number of entries (events) in the file

Data type of variable

Dimensions

`*` : means one entry per event

`* var` : means a vector with variable length per event (*ragged*)

(if just a number (e.g. `4`) it means a vector with the same length (4) for all the events)

Studying the ragged arrays

- Looking at one of the vector variables (*lep_pt*)

```
[14]
✓ 0s
    # Look at the first 4 (of 11458) events
    lepton_pt[0:4]

    [[43.8, 33.6, 31.2, 21.6],
     [20.9, 10.6, 11.2, 13.4],
     [69.4, 14.1, 39.6, 28.1],
     [130, 93.1, 22.7, 21.2]]
    --------------------------
    backend: cpu
    nbytes: 184.2 kB
    type: 4 * var * float32
```

axis = None: combines all values from the array into a single scalar result

axis = 0

Index	A	B	C
0	[#,#,#] →	[#,#] →	[#] →
1	[] →	[#,#,#] →	[#,#,#] →
2	[#,#,#] →	[#,#,#,#,#] →	[#,#,#,#] →
3	[#,#] →	[#] →	[#,#] →

→ : **axis = 1**

```
[9]
✓ 0s
    lepton_pt = tree_bkg['lep_pt'].array()
    # Counting...
    print("Number of rows = ",len(lepton_pt))
    print("Number of lepton in each row = ", ak.count(lepton_pt, axis=1))
    print("Total number of leptons = ", ak.count(lepton_pt, axis=None))
    print("Average leptons per event (multiplicity) = ",ak.count(lepton_pt, axis=None)/len(lepton_pt))

    Number of rows =  11458
    Number of lepton in each row =  [4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, ..., 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]
    Total number of leptons =  46039
    Average leptons per event (multiplicity) =  4.018065980101239
```

Content in data and MC

- The content, in terms of branches, is identical in data and simulation
  - But some variables might be empty or have standardized values in data
    - Truth collections are all empty in data
    - All scale factors are 1 in data

Truth electron p_T in data

```
[12]
✓ 0s
tree_data['truth_elec_pt'].array()[:4]

[[],
 [],
 [],
 []]
----
backend: cpu
nbytes: 40 B
type: 4 * var * float32
```

Truth electron p_T in MC

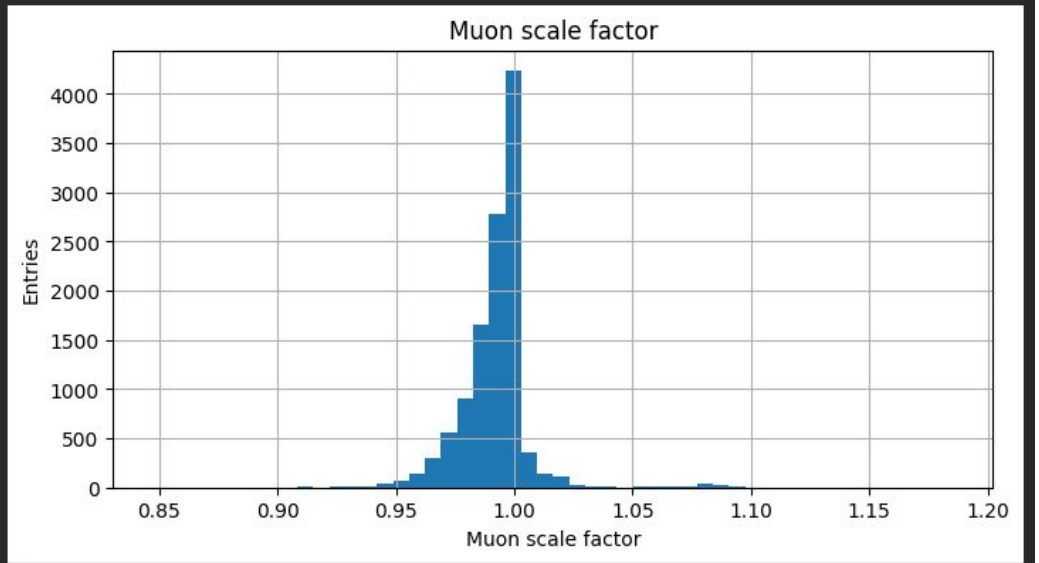
```
[13]
✓ 0s
tree_bkg['truth_elec_pt'].array()[:4]

[[44.5, 28.6, 50.8, 19.6],
 [10.8, 33.2, 13.1],
 [68.7, 14.4],
 [20.4, 126, 22.4, 90.7]]
--------------------------
backend: cpu
nbytes: 84.2 kB
type: 4 * var * float32
```

Look at the variables

- It is very easy to study the variables in the NTuples by plotting them directly from the array

```
[16]
✓ 0s
```

```python
import matplotlib.pyplot as plt

data = tree_bkg['ScaleFactor_MUON'].array()

# Create a simple plot
plt.figure(figsize=(8, 4))
plt.hist(data, bins=50)
plt.title('Muon scale factor')
plt.xlabel('Muon scale factor')
plt.ylabel('Entries')
plt.grid()
plt.show()
```



Look at the variables

- It is very easy to study the variables in the NTuples by plotting them directly from the array
  - But if you plot from a vector you need to flatten it first

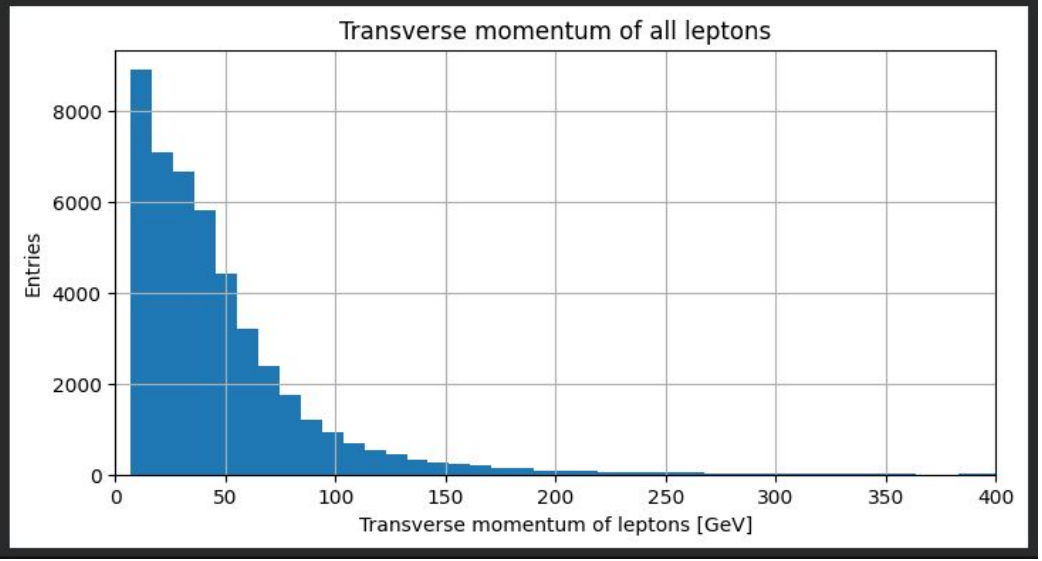
```
[21]
✓ 0s
tree_bkg['lep_pt'].array()

[[43.8, 33.6, 31.2, 21.6],
 [20.9, 10.6, 11.2, 13.4],
 [69.4, 14.1, 39.6, 28.1],
 [130, 93.1, 22.7, 21.2],
 [46.7, 60.3, 32.2, 39],
```

```
[19]
✓ 0s
ak.flatten(tree_bkg['lep_pt'].array())

[43.8,
 33.6,
 31.2,
 21.6,
 20.9,
 10.6,
 11.2,
 13.4,
 69.4,
 14.1,
 ...,
```

Look at the variables

- It is very easy to study the variables in the NTuples by plotting them directly from the array
  - But if you plot from a vector you need to flatten it first

```
[16]
✓ 0s
import matplotlib.pyplot as plt

data = ak.flatten(tree_bkg['lep_pt'].array())

# Create a simple plot
plt.figure(figsize=(8, 4))
plt.hist(data, bins=100)
plt.title('Transverse momentum of all leptons')
plt.xlabel('Transverse momentum of leptons [GeV]')
plt.ylabel('Entries')
plt.xlim(0, 400)
plt.grid()
plt.show()
```



Scale the background to the right luminosity

- Need to calculate the total weight from the formula for every event

$$w_\sigma = \frac{\int L dt\, \sigma\, \eta\, k\, w}{\sum_i w_i}$$

Loads the variables we need and iterates over all the events in chunks (see: uproot.iterate)

Collects all the events in a list before we convert it back into an awkward array at the end

```
[67]
✓ 0s
    sample_data = []
    # cross section is in picobarn, need to convert
    # the total integrated luminosity of 36.1 inverse
    # femtobarn to similar unit
    lumi = 36.1*1000
    # Define the variables we want to read.
    variables = ["xsec","kfac","filteff","mcWeight","sum_of_weights","lep_pt"]
    # Iterate over all the entries in the TTree. If no list is specified all branches are loaded!
    # It iterates over the entries in chunks. Default is to limit the memory to 100MB
    for data in tree_bkg.iterate(variables,library="ak"):
      # Calculate the weight and add it as a new variable
      data["totalWeight"] = (lumi*data["xsec"]*data["kfac"]*data["filteff"]*data["mcWeight"])/data["sum_of_weights"]
      # Append this chunk of events to the final output
      sample_data.append(data)
    # turns sample_data back into an awkward array
    background_diboson = ak.concatenate(sample_data)
```

Scale the background to the right luminosity

- Then plot with and without weights

The variable we want to plot and the weights to apply for each event/entry in the vector

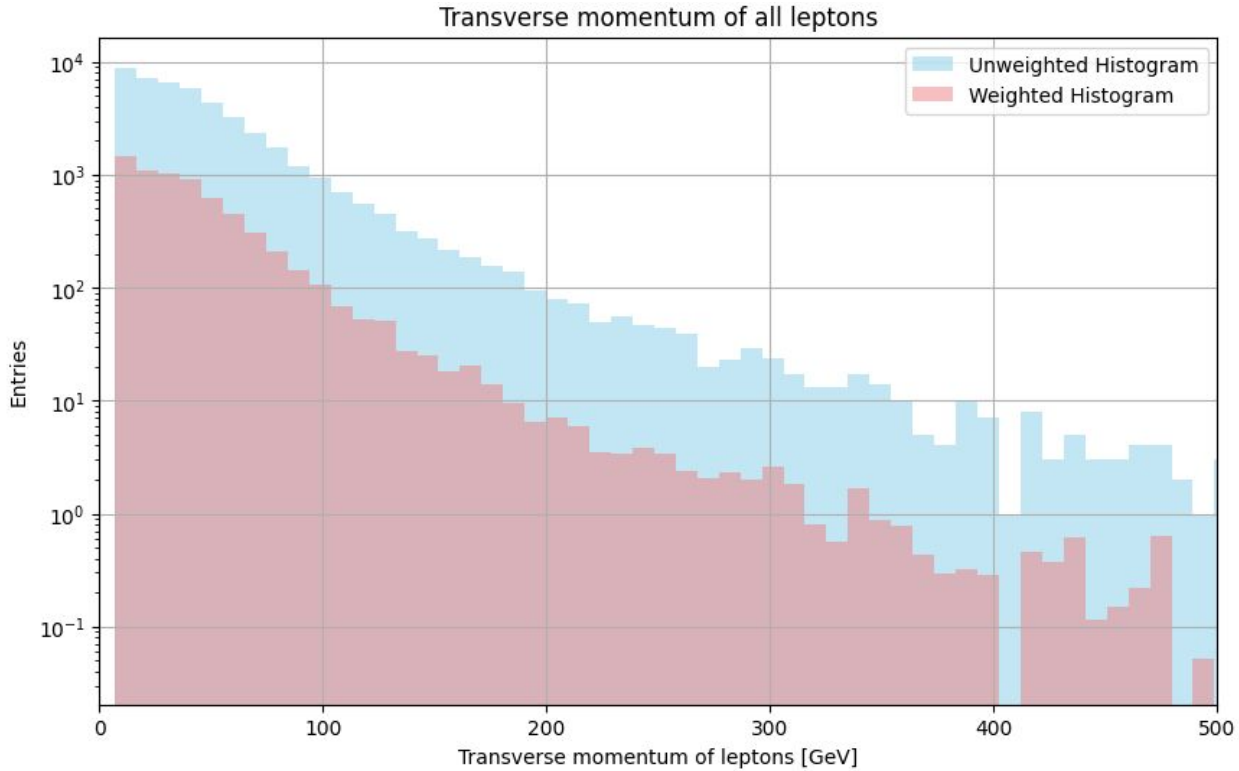
```
[69]
✓ 0s
mc_weights = ak.to_numpy(background_diboson["totalWeight"])
data = background_diboson['lep_pt']

# Expand event_weights to match the flattened data
# This broadcasts the event weight to all elements within that event's list
expanded_weights = ak.flatten(mc_weights * ak.ones_like(data))

# Flatten the data itself
flattened_data = ak.flatten(data)

fig, ax = plt.subplots(figsize=(10, 6))

# Create a simple plot
plt.figure(figsize=(8, 4))
ax.hist(flattened_data, bins=100,alpha=0.5, label='Unweighted Histogram', color='skyblue')
ax.hist(flattened_data, weights=expanded_weights, bins=100, alpha=0.5, label='Weighted Histogram', color='lightcoral')
ax.set_title('Transverse momentum of all leptons')
ax.set_xlabel('Transverse momentum of leptons [GeV]')
ax.set_ylabel('Entries')
ax.set_xlim(0, 500)
ax.set_yscale('log')
ax.grid()
ax.legend()
```

When we flatten the data we need to expand the weight vector accordingly

Provide the weights when creating the histogram

Scale the background to the right luminosity

- Final plot!



Time for Q&A!

Questions?